Universidad Carlos III de Madrid

Bachelor Thesis

A Client-Server Architecture for Distributed and Scalable Multimedia Content Analysis: An Android App for Assisting Phone Users in Shooting Aesthetically Valuable Pictures

---

Author

Gabriel García Ardiles

Advisors

Fernando Fernández Martínez & Alejandro Hernández García

Degree in Biomedical Engineering

Leganés, June 24th 2016

Abstract

Nowadays developing modern scientific image and video analysis algorithms faces the issue of distributing them among the open community with multiple versions for very different platforms. This requires software development skills usually unknown by the researchers outside of the computer science world. Client/server communications have acquired a leading role by abstracting the business logic of applications from thin clients running on small devices like smartphones which end users can carry with them.

The present work describes the design, modeling, development and testing of a client/server architecture that has the ability to perform computations on image and video characteristics on independent Matlab® instances and offer production efficient SQL persistence to store the results. All of this, immersed in a user authenticated environment. This project has been specifically focused on a currently ongoing study by researchers from Universidad Carlos III and Universidad Politécnica de Madrid. Their main goal is to estimate the aesthetic value of images and videos by the computation of audiovisual content. However, the architecture has been designed and built with the objective of being applicable to any kind of biomedical, audiovisual or any other engineering image or video analysis study.

Resumen

Hoy en día, desarrollar nuevos algoritmos científicos que analicen videos o imágenes lleva consigo el problema de la distribución abierta a la comunidad con las múltiples versiones de las distintas plataformas utilizadas. Para que ello sea posible, se requieren habilidades de desarrollo de software que normalmente son desconocidas por parte de los investigadores no inmersos en campo de la informática. Las plataformas cliente/servidor han adquirido un rol primordial al abstraer la funcionalidad principal de las aplicaciones de los clientes livianos como los teléfonos inteligentes que pueden llevarse en el bolsillo.

Este trabajo describe el diseño, modelado, desarrollo y prueba de una arquitectura cliente/servidor que tiene la habilidad de realizar cálculos de características de imágenes y videos en instancias independientes de Matlab® y ofrecer persistencia de datos SQL al nivel de un entorno de producción donde guardar los resultados obtenidos, todo ello sumergido en un ambiente donde los usuarios están completamente autentificados. Este proyecto ha estado particularmente enfocado a una investigación actualmente en desarrollo por investigadores de la Universidad Carlos III y la Universidad Politécnica de Madrid. Esta investigación tiene como objetivo el estudio del valor estético de imágenes y videos a través del cálculo de descriptores objetivos. De todas maneras, la arquitectura se ha diseñado y construido con el objetivo de posibilitar la aplicación a cualquier otro estudio dentro de la ingeniería biomédica, audiovisual u otra ingeniería donde se requiera el análisis de video o imagen.

Acknowledgement

I wish to acknowledge my mentors, Fernando Fernández Martínez and Alejandro Hernández García, for their assistance through all the research and development process. Their help and open mind towards proposed solutions has been of great importance. Also, their patience when explaining the main aspects of the Matlab descriptors implementations has been crucial for the evolution of this project.

I am specifically grateful to Juan José Vaquero López and Javier Pascau González Garzón for their research on the requirements of a bachelor thesis and the interest they have shown on a software project design that I was building coming from a biomedical background.

I would also like to thank my friends and family for their support and willingness to help. Special thanks to my mother for her unconditional love and revision outlooks; to Marta Sondesa Polo for revising the grammar of this document and reviewing the contents of the project; and to everyone else that has helped me with the redaction of the document.

“Have the courage to follow your heart and intuition.
They somehow know what you truly want to become”
Steve Jobs

Table of Contents

List of Figures

List of Tables

1. Introduction

Beginning in the 1980s, computers successively started to become an essential element in practically every industry. The Information Revolution just started shifting the work routines of the entire world. As doctors in economic history from Oxford and Sweden explain, computers began to "shift the composition of employment" by drastically removing "many middle-skill routine jobs" and augmenting the demand for "abstract jobs" as well as creating new job positions [1]. After the 1990s, a new communication concept started to gain weight in the computer industry: Internet. The World Wide Web became a reality which boosted productivity of businesses and laid the foundations for client/server implementations which would acquire outstanding relevancy in the following ten years. Professor Daniel E. Sichel defines this process as an "Information Superhighway" where an exponentially increasing number of "households and businesses were buying more and more powerful computers and hooking them up to the internet" [2].

Over the course of the last years of the 90s decade and the first years of the 2000s a huge amount of internet-based companies started their business. It was the dot-com era, as defined by authors from the International Monetary Fund [3], where a lot of companies started to grow in a process known as "dot-com bubble" followed by, unpredictably by the majority of people, a huge crash known as the "dot-com bust" [4]. This period was characterized by the introduction of software development processes which although still not optimized did set the basis for continuous deployment of development systems like the Agile Software Development Model [5].

Nowadays, in 2016, software is all around our lives. According to the independent market researcher *eMarketer* in 2016 there will be "2 billion smartphone users in the world" [6]. The fact that the vast majority of people has a smartphone revolutionizes the way software should be built. Mobile devices are the preferred distribution target for newly-developed software. Therefore, the use of specific architectures that permit the abstraction of complex computations (i.e.: cloud computing and client/server architectures) gain a remarkable importance.

The evolution from the computer revolution to the internet era has set the foundations for what this project's architecture will be. The design has focused on a client/server system where computation is abstracted away from a cross platform mobile client leaning on the concept of cloud computing. This will permit running a complex set of Matlab® algorithms which could potentially analyze audio, gestures, images and videos from a mobile phone in an authenticated environment. Distribution of newly-developed pieces of scientific software can be easily distributed using the architecture proposed and built in this project, called from here on AffectivePixels architecture.

1.1. Terminology

The specific terminology used in this bachelor thesis is explained in detail in table 1. For the convenience of the reader the terms have been ordered alphabetically.

Table 1 - Terminology in alphabetical order

Term	Description
Agile	Software Development Model promoting planning and reiteration
Angular JS	JavaScript Model View Controller framework to create web applications. Ionic is based on Angular.JS.
API	Access Point Interface. The available functions to be called from a piece of software.
Cordova Plugin	Native code wrappers that can be called from JavaScript (i.e.: Camera, Network Information, GPS)
CXF	Java web services framework
Django	A Python framework to build web applications
DRF	Django Rest Framework. A Python framework based on Django to build REST web services.
Facebook	Social application available online
FFmpeg	Cross-platform solution to record, convert and stream audio and video [7]
GitHub	A Git version control repository opened to the public.
HTTP	Hypertext Transfer Protocol. Communication protocol used in the World Wide Web
Ionic Framework	JavaScript cross-platform application development framework that can export to Android, iOS and Windows Phone

Java	Programming language
Matlab	Scientific programming language
Matlab Instance	A computing object that can execute Matlab commands. It is opened in a different window and can maintain communication through a ZMQ socket
MatlabControl	Java library to launch and connect to Matlab instances
Mixin	A preprogrammed template that covers a widely used piece of functionality and can be imported to custom functions to save time.
NodeJS	Non-blocking I/O, event based and JavaScript based programming language that runs on Google's V8 Engine
NPM	Node Package Manager. Dependency control for NodeJS programming language
ORM	Object Relational Mapping. Relation between objects in a programming language and SQL Database Tables
PaaS	Platform as a Service. It is a modern cloud service that offers automation on other aspects different from the application like deployment, server setup, shutdown recovery...
Python	Object oriented interpreted programming language
REST	Representational State Transfer. It is a convention of the API endpoints to be more representative of the resources offered.
Roadmunk	Online roadmap design service
Spring Framework	Java dependency injection framework that simplifies the nexus between modules of a large application
UX	User experience. A field of software design that studies the interaction of the application with the user.
WhatsApp	Social chat application available on mobile devices

2. State of the Art

Prior to developing an efficient software architecture which ought to fulfill multiple requirements it is of the outmost importance studying the implementations and improvements in the field of cloud image and video analysis at the present moment. Chapter 2 explains many related architectures that solve problems of similar nature to the one proposed in this final thesis. The products and architectures analyzed in this section are not supposed to compete with the AffectivePixels architecture but serve as a reference checkpoint to ensure the work is always heading towards bleeding-edge technology.

2.1. Cloud Computing on PaaS

One of the most important software technology breakthrough of the last decade has been the platform as a service (PaaS) architecture. This type of service provides developers with a ready-to-go platform to avoid the hassles of maintaining a complex server architecture. Depending on the control of the system that is required by the developer different cloud services could be used. In figure 1 three of them are presented. As it can be observed, infrastructure as a service (IaaS) automates less tasks than PaaS and provides the possibility to exert a higher control on the upper layers of a system. However, PaaS bears the most interesting relation of control vs automation for a cloud computing environment where custom functionality has been programmed by the developer. In contrast, software as a service (SaaS) provides an on-demand ready-to-use software applications with no developer work, i.e.: Google Docs. Nowadays, there are different commercial solutions available to the public that offer these type of services: Azure, Amazon Web Services or Google App Engine among others.

Implementations leveraging on PaaS architectures gain rising importance when targeting the mobile market leading to inexpensive web applications exposed to the public for a distributed use with automated functionality. Researchers from NTU rephrase mobile computing as an “infrastructure where both the data storage and data processing happen outside of the mobile device” where a simplified environment reports multiple advantages specifically by providing out-of-the-box “building, testing, and deploying custom applications” [8]. Diving deeper into image and video analysis, other authors have researched on the benefits of PaaS parallel video analysis in the cloud [9] proposing a “unified framework” that can “integrate and configure video analysis engines” within a “user-specified workflow”. This last framework evidences a highly similar aim to what the AffectivePixels architecture should do.

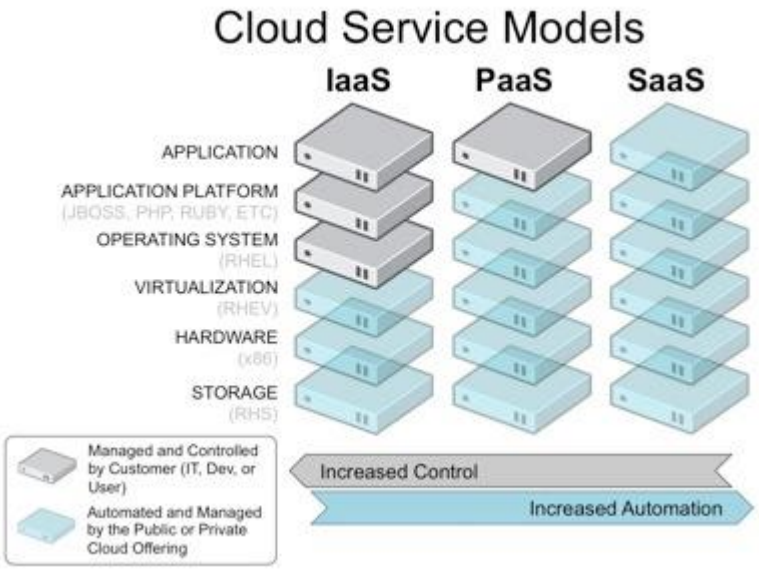


Figure 1 - Cloud service models. IaaS, PaaS and SaaS.

The PaaS cloud model is a less expensive solution to consider for the AffectivePixels architecture when ready for a production environment. It will reduce costs to an on-demand basis and system capabilities could be increased dynamically as well. Also, the free tier of services like Amazon Web Services could permit a research team with limited resources expose their new algorithms without any cost for at most 12 months [10].

2.2. Google's Computer Vision API

Another significant online service that can be considered state of the art technology on audiovisual analysis is Google's Vision API [11]. This self-contained framework opens a whole set of image analysis endpoints that a client application can consume by common HTTP request. An example call to the online API can be seen in figure 2 where the animal species, subphylum and even the breed of the dog is extracted by the framework. This type of advanced analysis algorithms are suitable for used on small and less powerful devices like a smartphone or even a smartwatch. Computer vision and pattern recognition as explained by the World Scientific Handbook has "now reached maturity" and "will definitely play a very major role" in the upcoming years [12]. The latter, studies "mathematical techniques such as statistical techniques, neural network or support vector machine" to detect and classify patterns [13]. Pattern recognition is used by computer vision which according to Ballard and Brown has the mission of creating "explicit, meaningful descriptions of physical objects from images" [14].


```
{
  "responses": [
    {
      "labelAnnotations": [
        {
          "mid": "/m/0bt9lr",
          "description": "dog",
          "score": 0.97346616
        },
        {
          "mid": "/m/09686",
          "description": "vertebrate",
          "score": 0.85700572
        },
        {
          "mid": "/m/01pm38",
          "description": "clumber spaniel",
          "score": 0.84881884
        },
        {
          "mid": "/m/04rky",
          "description": "mammal",
          "score": 0.847575
        },
        {
          "mid": "/m/02wbgd",
          "description": "english cocker spaniel",
          "score": 0.75829375
        }
      ]
    }
  ]
}
```


Figure 2 - Google Vision API. Sample implementation where the animal, subphylum and breed of a dog is detected. Source: Google Vision API: https://cloud.google.com/vision/docs/quickstart#make_a_request_to_the_cloud_vision_api_service

The concept of a rest API accessible through HTTP calls is a valuable concept to be included in the AffectivePixels design. The identification key approach to authenticate and gain access to the services should also be introduced in the implementation for a production-ready environment. Overall, Google's vision API brings users with the concept of abstracted online implementation and a similar idea should be broadcasted to the work performed in this project.

2.3. VMAF

Video Multi-Method Assessment Fusion is a video analysis tool whose goal is to evaluate the quality of the video files according to the human observer and not by specific measurable pixel properties of the video. As Liu, Tsung et al. explain it is a "nonlinear combination of scores from multiple methods using a set of suitable weights obtained by a training process" [15]. This statement explains that the main methodology for this algorithm is the evaluation of a regression expression whose coefficients are obtained through a machine learning procedure. These set of coefficients or weights are not

straightforward to compute from a programming perspective since they are somehow subjective and a large set of cases has to be analyzed to infer a pattern.

A new VMAF implementation has been recently launched [16], in June 2016, by Netflix, Inc. to improve their streaming service video quality. The company has opened their source code to the community of developers and is available on GitHub [17]. Their custom implementation is based on three elementary descriptors as explained on The Netflix Tech Blog [16] :

- Visual Fidelity: Measure of information fidelity loss using a modified version of the four scale metric.
- Detail Loss Metric: Measure of the loss of details affecting the content visibility and also the redundant impairment that tends to distract a viewer attention.
- Motion: Temporal difference between consecutive frames of a video.

The concept that the VMAF algorithm solves, i.e.: video perception analysis for the human viewer, is one of the main objectives of the AffectivePixels current research. The combination of scores to obtain a classification method is a concept to be introduced in a final classification descriptor which combines other descriptors after a machine learning process in a nonlinear regression model.

2.4. Summary

As a final assessment of the previous state of the art it is necessary to extract the main ideas that should be included in this work in order to have a competitive technology. The PaaS model should be the desired architecture to use at the moment of final deployment. Therefore, the architecture should be built in such a manner that is easily deployable with a small amount of simple commands needed to deploy to a Django-ready server in AWS, Heroku or Azure for instance. The server should expose a REST API where the computation and business logic is completely hidden just like Google's Vision API. It should also return a JSON response since it is a flexible structure that keeps a loose connection with the client permitting a wider set of client platforms. Additionally, the client token sent with the request to Google's Vision API is an ideal authentication technique to be included in the system. Finally, the VMAF algorithm is a great idea of what a final classification descriptor should be like. A regression model improved by machine learning processes could be used to classify an image and evaluate its aesthetic value.

3. Methodology

This chapter describes the principal aspects that define the project. A brief explanation of the problem that needs to be solved is covered in the first section. The motivation of the writer to work on this problematic is explained next. Then, the objectives and milestones set by the mentor and verified by the student to be accomplished. Finally, the last section introduces the final resolution method to the problem which underwent the design, development and testing phases covered in chapters 4 and 5.

3.1. The Problem

The work in these project was born tied to a current research that attempts to explain the aesthetical value of audiovisuals as perceived by human observers with the computation of objective descriptors. Ideally, a simple distribution of the newly-developed image and video analysis software to the mobile market was desired. These algorithms were implemented in Matlab® for simplicity purposes on the development process but the end distribution was expected to be on mobile devices to enable rapid user growth. There was a need for a ready-to-go solution for the distribution of a platform-specific software to the cross platform mobile market without the use of Matlab® translation packages and the ad-hoc development to either Android or iOS. These libraries (Matlab Coder® and Matlab Compiler®) are very prone to errors and do not support all the methods available in the different Matlab Toolboxes. The portability is not fully ensured and very time consuming since it requires an advance knowledge of computer science skills to adapt the generated code to the specifics of Android (Java programming) and iOS (c-objective and Swift programming).

3.2. Project Motivation

It has been clear in the last decade that software has reached a drastic importance in engineering fields all-around. As a future biomedical engineer, I have embraced these technologies and refocused skills towards the use of software for the solution of broader engineering issues. I have always had medical concerns in mind building an architecture that incorporates use cases of the biomedical field as well.

I have driven my work specifically towards the solution of a currently ongoing research study: “measuring the aesthetic value of images and videos by objective descriptor values" [18] [19]. However, the architecture proposed could be used for any other image and video analysis software (i.e.: a cell counting algorithm or a flux measurement tool).

3.3. Objectives

Researchers who develop novel scientific algorithms that modify current mathematical implementations are required to know complex concepts and empirical or theoretical laws of their specific field of study. Expert engineers in energy, biomedical, aerospace or any other degree not focused on the understanding of computer science are not usually taught software development skills that would let them distribute their innovations across the multiple platforms available in the market.

The work presented herein has the goal of implementing a client/server architecture solution that works platform independently. The system should be easily configured by console commands and provide multiplatform support by exposing HTTP endpoints accessible from the internet. Any client with internet access should be able to make a request to those endpoints and receive common JSON objects as a response. A sample mobile client application has also been built for demonstration purposes.

The main advantages of this design will be:

- Unneeded ad-hoc development: The algorithms can be designed independently of the destination platform. There is no need for translation packages, compilation packages or specific knowledge of the custom parts of the native platforms.

- Matlab Pool: The server launches a pool of computing instances ready to execute computation. A balancer distributes the load equally among them. By just changing a parameter, the administrator can control the amount of computing units to be launched. This translates in computation speed but also in a larger footprint on the server.

- User authentication: The server leverages an authentication and permissions system out-of-the-box. A simple console has been included to let the administrator create and modify new accounts and also control the server resources.

- Flexible computation storage: Computation results may have very different shapes as there are many available data structures to use in the various programming languages. This heterogeneity will be addressed by translating any incoming data structure to its JSON representation and storing the string on the database. The recovery will be as easy as it is parsing the object.

- Dependency management: The descriptors may have dependencies on others. This is the case, for example, of a classification descriptor that may depend on the colorfulness, intensity and entropy values. These dependencies are resolved by the server automatically. The only requirement is to set them in the administrator consoles and follow the programming guideline in annex III.

- Administrator console: The administrator doesn't need to access the code in order to control the server. There is a console available to visually make all the changes without programming a single line of code. The console is available by navigating to the url:

  http://affectivepixels-server.com/admin/

3.4. Resolution Method

A client/server architecture design and development has been the platform independent solution desired to efficiently distribute a set of image processing and computer vision algorithms. A Django server running on the cloud will be in charge of all the Matlab computations, user authentication and permissions, image and video storage, indexing descriptor computation results in a MySQL database and providing an administration console for server control and maintenance. Also, a mobile demonstration application has been built using the second version of the Ionic Framework which is a JavaScript framework that can be exported to both Android and iOS platforms.

4. Architecture Design Process

This chapter depicts the temporal milestone roadmap for the design of an end-to-end image and video analysis Client / Server System. It will guide the reader through the various hypothetical architectures designed taking as a reference some of the state of the art technology discussed in chapter 2. These proof of concept designs have been implemented in a simplified manner for testing purposes in order to have a better perspective for the selection of the final design. A discussion on the various technology requirements will also be included in this chapter and written according to the Standard-830 as set by the IEEE [20].

4.1. Roadmap

The time and effort applied on the specific parts of the project has to be deeply optimized to meet the delivering requirements. That is the main reason for a roadmap to be utilized in any engineering project. The planning for the AffectivePixels client/server architecture has followed three consecutive steps closely matching the initial design steps suggested by the Agile Software Development Model [5]. A general Agile SDM is shown in figure 3. As we can see, for this project the main focus was on the central development region marked on blue. The work has never been focused on continuous delivery of the software. That part is a future mechanism to ensure efficient detection of errors and efficient updates and improvements.

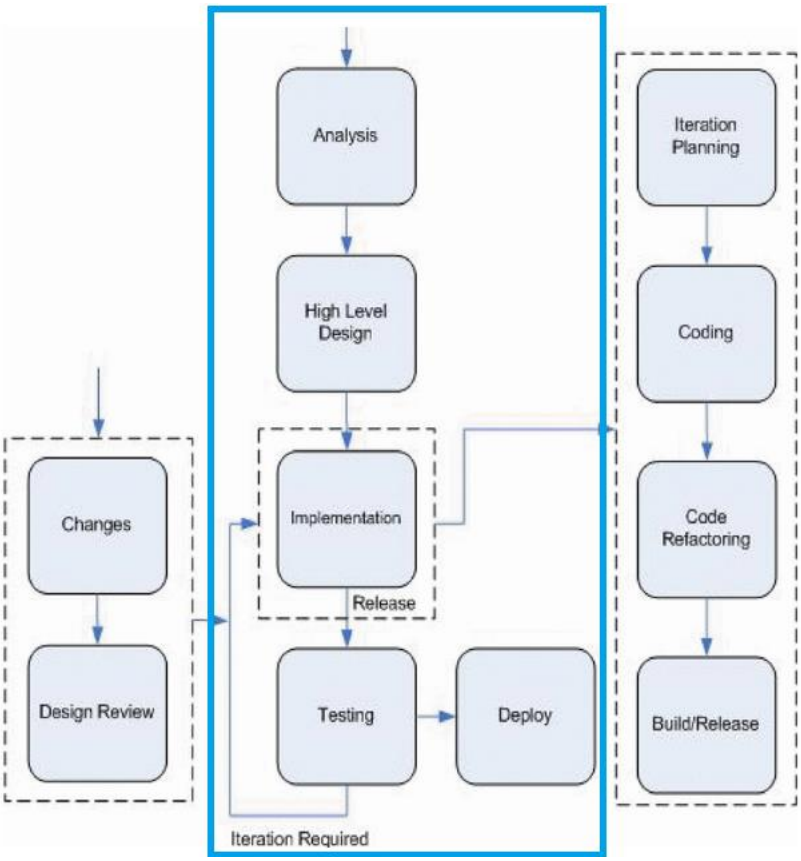


Figure 3 - General Agile Software Development Model. Source: Figure 4 in "Agile software development: Impact on productivity and quality" [5].

The three steps of the AffectivePixels roadmap can be observed in figure 4 which is a screenshot of the *roadmunk* application, an online roadmap developing service. The first period of the project roadmap has been a research phase that lasted almost one quarter of a year. For these two to three months, the possible technologies to be used were all deeply studied, looking at technical documentation, community examples and usage experience by larger companies. In order to make a good selection beforehand some application samples with a simplified functionality were built and tested to verify correct applicability to the problem. Once the technology was finally selected, after the 10 days deciding period, the development phase began. During this period, the different characteristics of the system were assigned a time label on the roadmap when they will be worked on. These were general steps that were expected to change along the way but also serve as a general guidance that ensured the delivery would be on time. Finally, the third step, was a testing phase where the already fully built system was subjected to extensive testing and was expected to fail. A reiteration process was then initiated were the corrections were included and testing redone. This process ended in the first two weeks of June as expected.

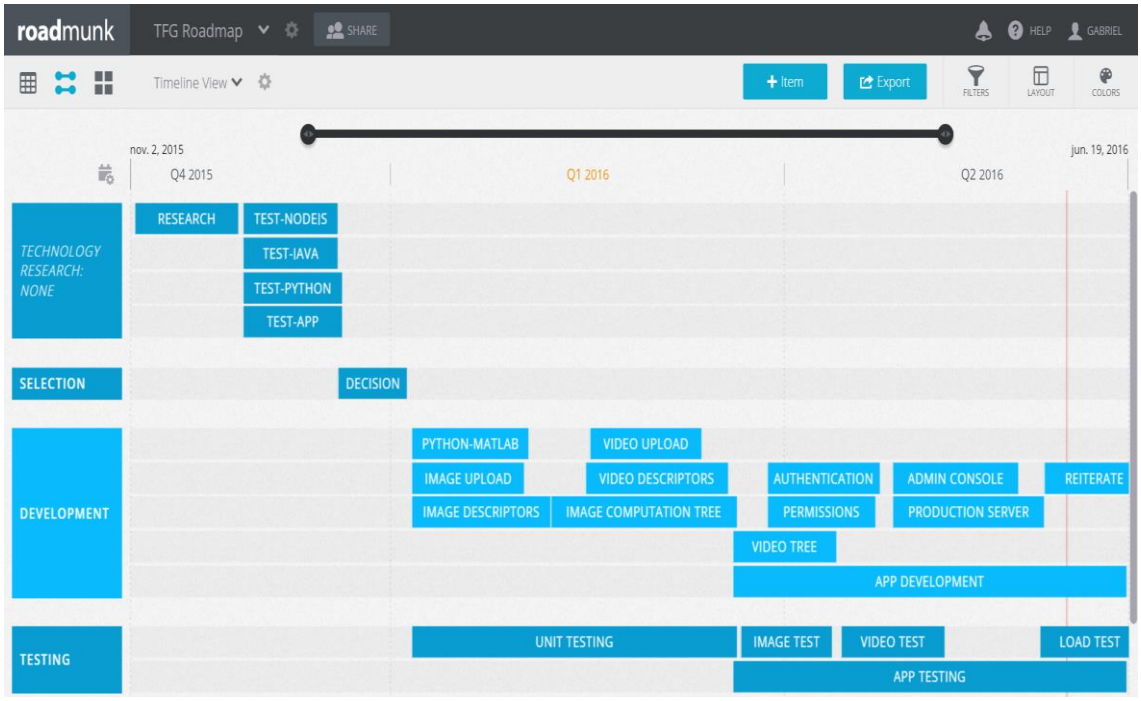


Figure 4 - Roadmap of the Architecture development

4.2. Research Phase

A lot of documentation was read on this period about the different technologies used to develop web services. Out of all, the top three candidates according to the project requirements were selected: Java, Node.JS and Python. Then, simplified applications were built for each one of these three programming languages to better assess their applicability to the project. In the end, the Decision Matrix shown in table 2 was built. This table summarizes the pros and cons of each technology for the development of the AffectivePixels client/server architecture. For each characteristic there is grade of 1 to 3 depending on how much it benefits the project. At the bottom of the table, a sum of all the grades is shown and also a normalized grade for each technology taking into account that 39 is the maximum and presented as percentage numbers.

The most important decision characteristics where these:

- Implementation time: Temporal estimation of the time it took to code the same sample functionality across the different frameworks.

- Read capability: Easiness to give information about the different classes implemented and how can another programmer understand the old code.

- Available documentation: Amount of online resources that could be found with technical information about the API and open community size estimation.

- Quality of the documentation: Relation between the easiness of the documentation reading with respect to the level of technical detail present in the text.

- Latency: Amount of time since an HTTP Request was sent and the Response was received for the same Matlab function.

- Environment required: Assess the amount of requirements to have the system running on a particular machine.

- Concurrency: Type of programming technique to use depending on process handling and threading capabilities of the language.

- Testing frameworks: Integration of testing libraries that make automatic unit tests easier to develop without too much overhead of class instantiation.

- Object Relational Mapping: Evaluate the existence and ease of implementation of an Object / SQL database mapping.

- Security and authentication: Assess the level of security available with a simple implementation for non-shared resources within the application.

- Integration with front-end: Easiness of communication between the front-end JavaScript, specifically JSON objects, and the server side programming language.

- Matlab connection: Evaluation of the way the technology could connect to Matlab in the easiest way possible.

- Plug and play functionality: Assess the existence and volume of preexisting implementations of functionality needed in the AffectivePixels architecture.

Table 2 – Programming language Decision Matrix

	Java	Node.JS	Python
Implementation time	1	3	3
Read capability	2	2	3
Available documentation	2	2	3
Quality of the documentation	1	3	2
Latency	3	2	3
Environment required	2	2	3
Concurrency	3	2	3
Testing frameworks	3	2	2
Object Relational Mapping	2	2	3
Security and authentication	2	2	3
Integration with front-end	1	3	2
Matlab connection	2	1	3
Plug and play functionality	1	3	3
Sum	25	29	36
Normalized (%)	64.10	74.35	92.30

4.2.1. Java

It is an object oriented, concurrent language that has been in the software field since 1995 [21]. It has a large community with a lot of documentation and stable frameworks. For this project a first sample server application was implemented using the CXF [22] and Spring Frameworks [23]. These are used for web service development and dependency injection in java programs.

The first testing version using java was able to launch a pool of independent Matlab instances capable of doing work. The library used to do so was *MatlabControl* [24]. Then, with each incoming computation request, the Java program would connect to each one of the Matlab instances, one by one, to get a response. This concept is known as *load balancing*. As computer scientists say: "In the cloud computing paradigm, load balancing is one of the challenges" [25]. This concept is largely used in cloud computing because heavy computations have to be done by a limited amount of resources. It's not 10 computers running their Matlab programs it's 10 computers asking a single remote one to do all the work.

After testing this sample implementation there were some advantages observed like robust multiplatform support, seamlessly running on Windows and Linux operative systems, concurrency was one of its best capabilities as well as the Junit [26] testing library which made writing automatic concurrent tests easier. However, these advantages couldn't overcome the large drawbacks: Java is a very time consuming and strongly typed language. It doesn't really support out-of-the-box functionality and its documentation is not well written. These disadvantages make Java the worst choice out of the three for this single-developer final thesis project.

4.2.2. NodeJS

It is a very complete JavaScript-based server side development language. NodeJS [27] runs on a single thread inside Google's V8 JavaScript Engine. It is a modern language built in 2009 and benefits of many new capabilities like JavaScript event-driven programming which lets the developer spread a message across the listeners within the application, the non-blocking I/O which makes it very light on the computer or the package manager NPM [28] to install dependencies.

The first testing program developed using NodeJS was very lightweight. With a simple command run in the console: *npm init,* the whole project structure is built. The main advantages of this technology were the easiness of dependency injection, where every needed module is easily imported through an npm command, the simple express.js framework lets the developer build REST API very fast and start up the server with a few lines of code. Also, NodeJS had a very light footprint on the server since it runs on a single thread. This technology is considered to be non-blocking because it asynchronously runs its tasks. There is a process in NodeJS known as event loop that stops any blocking execution and starts some other while the first one finishes. That way there is no time lost in blocking operations, like for example awaiting an HTTP response from a third party server, or read / write operations to the hard drive (i.e.: saving a jpeg image to the hard drive).

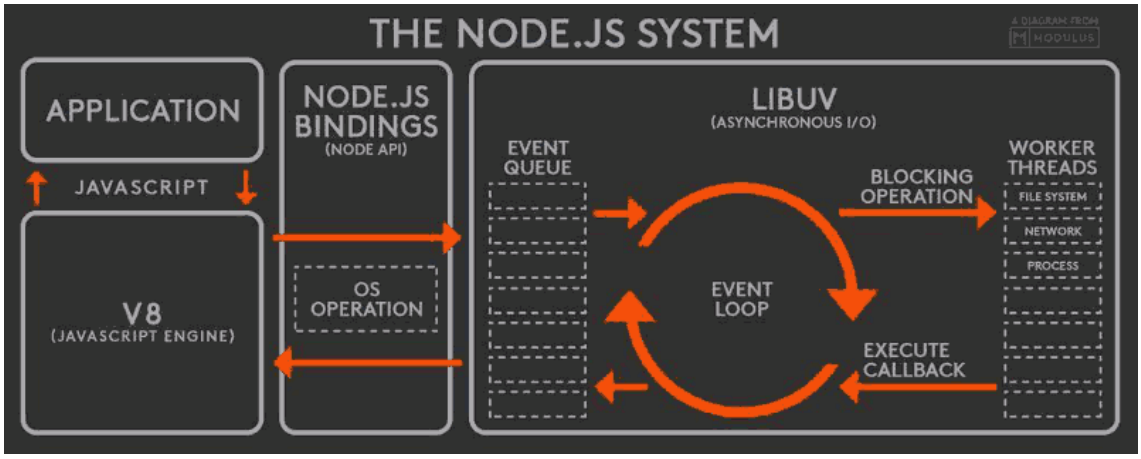


Figure 5 - NodeJS Event Loop Diagram [29]

NodeJS seemed like a great option until the connection to and pooling of Matlab instances was implemented. The final implementation included a java connector class that I had to implement which could take care of opening a Matlab pool. The main problem with this process is that NodeJS runs in the Google's V8 Engine which is very different from the Java Virtual Machine. This bridge between the two interpreters was a lot of overhead which could create unexpected issues. Another problem related to this was the code maintenance of two different technologies and also testing of the two.

4.2.3. Python

It is an interpreted language whose main focus is to provide very legible code to the programmers. Python [30] was built in 1991 is multiplatform, supports various paradigms and has dynamic typing. It is widely used in scientific programming with its various libraries such as *NumPy* [31] for vector calculations or *Matplotlib* [32] for scientific plotting. Python, like NodeJS has a very useful package manager known as *pip*. In order to program a web application there is a specific framework known as Django which I used for this project.

The testing application was developed using Django Rest Framework, a specific Django-based framework to develop web services. DRF is very self-contained, it has its own ORM, object relational database model, which makes accessing the persisting SQL database much simpler. It also has a strongly predefined structure, prevailing convention over configuration, which makes a project easier to be modified by future developers. By using

the Anaconda Python Distribution [33] the multiplatform use is enhanced. Also, the package manager simplifies a lot dependency injection. DRF has support for *mixins* which simplifies the development of common paths of a REST API. Very importantly as well, python has its own library to launch Matlab instances and communicate to them: *Pymatbridge* [34].

The disadvantages of using Python with DRF are the learning curve of their conventions, but once known development becomes easier in favor of architecture design, the key of the project. Apart from that, this technology was the perfect fit for the project.

4.2.4. Client Application

The research for the client app was done in a similar manner. A new decision matrix was built to quantify the applicability of each technology to the project. Every field had a numeric grade from 1 to 3. In this case, an important requirement was to have a cross platform application. In other words, coded once and shipped across different platforms with no effort: "Cross-platform is necessary in current mobile market" [35].

The main technologies that were researched were PhoneGap, Cordova, Ionic, and native programming and the different fields addressed were the following:

- MVC structure: Organization of the code inside a model-view-controller framework is a factor that speeds up development. In other words, the use of convention over configuration accelerates development.

- Multiplatform distribution: Ad-hoc developments should be avoided as a project requirement. A single-engineer project requires the use of technologies that permit the automatic compilation to other platforms from one single code base.

- Application styling: The easiness of client styling and availability of predesign templates is assessed in this field.

- Performance: Responsiveness of the application and ability to handle animations and transitions. Native implementations undoubtedly have the greatest performance.

- API access: Accessibility to the hardware. Most of the technologies are able to access common sensors like the camera or the GPS.

- Development IDE: This field assesses the available Integrated Development Environments for the different technologies.

- Community support: The size of the available experiences and help that can be received by the users of the technology.

- Documentation: The amount of available documentation and readability.

Table 3 – Cross Platform Technologies

	Cordova	Ionic	Native	PhoneGap
MVC Structure	1	3	3	2
Multiplatform distribution	2	3	1	3
Application styling	1	3	2	2
Performance	2	2	3	2
API Access	2	2	3	2
Development IDE	2	3	2	2
Community Support	1	3	3	2
Documentation	2	3	3	2
Sum	13	22	20	17
Normalized (%)	54.16	91.66	83.33	70.83

From the previous table it can be observed that the best technology to be used for this specific project is Ionic. In this case, the performance is lower but very similar to the native reference. An important point to consider is the available access to the camera which can be accomplished using a native plugin. In summary, the organization of the development structure and the overall maturity of the framework has made it the most applicable technology to the project.

4.3. Development Phase

For the planning of this period it was of outmost importance the control of the timeline in order to have an on-time delivery. The roadmap was divided on the specific functionality that the AffectivePixels server should have. Due to technical problems that typically arise on every development process the specific tag durations were modified on the course of the development. The

roadmap previously shown in figure 4 depicts the very last modifications. The first efforts were focused on getting image functionality to work. After that worked, the focus was directed towards video functionality. The reason will be better understood on chapter 4 where the model of the system is presented but mainly it is because videos will be decomposed into frames which can be treated as images. So having image functionality means also having frame functionality.

The client app was planned to be developed after there was some basic functionality already implemented on the server. The reason is the Client / Server Architecture. There will be no client functionality without a server. Since the app is so dependent on the server it was not until the last part of the roadmap when it achieves an important role.

4.3.1. First Architecture Version

This initial challenge was to get a working version where just images where uploaded. The server should be able to call Matlab and compute a descriptor by a URL query. In essence the client would make an http POST call like:

http://aesteassist-server.com/?descriptor=colorfulness

The POST request body included the image as a multipart binary field as well as other complementary data like the name of the file.

A complete diagram of the first architecture is shown in figure 7 (top). We can observe how the client made blocking requests to the server. If instead of one client there were several of them, the response times would have been increased sequentially.

Once the request arrived to the server, the attached image was saved to the hard drive. After that, the Matlab instance was called which also had access to the hard drive. Once the computation was done, the Matlab instance returned the result over to the server and the HTTP response made its way back to the client.

The presented architecture has to be considered as a proof of concept. The roundtrip was completed but there were no user authentication techniques, non-blocking behaviors nor any SQL indexing of the results.

4.3.2. Second Architecture Version

This second version was introduced to include the missing functionality from the first design. The video resource was included in the audiovisual model as a separate class than the image (figure 6). The frame entity was also added as a subset of the video. The image and video upload

and download functionality was separated from the computation API. Also, the descriptor model and API was introduced so that the client knew beforehand if the intended computation was available on the server side.

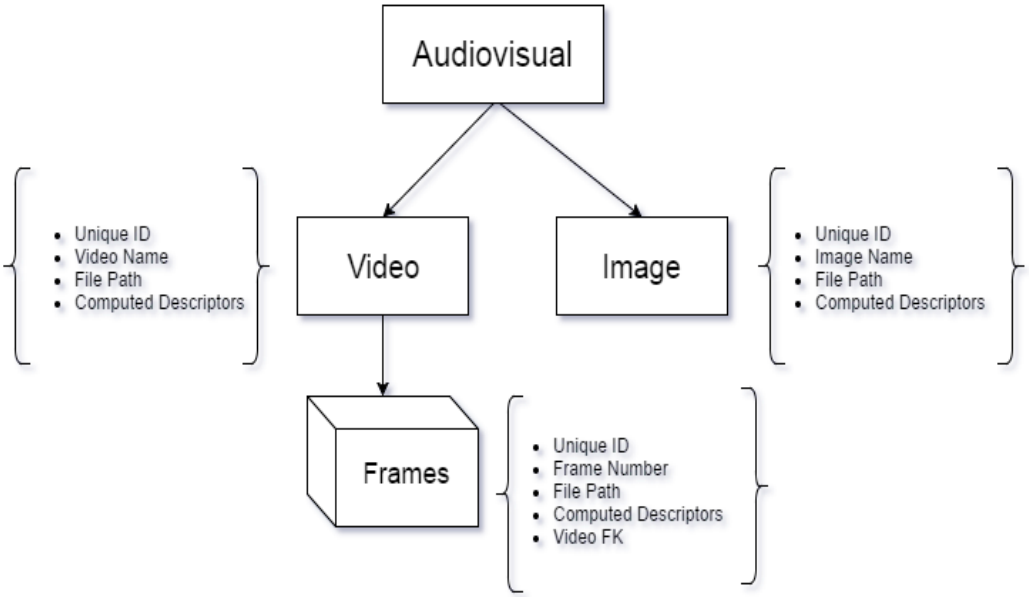


Figure 6 - Second architecture model

A diagram explaining this second version of the proposed architecture is shown in figure 7 (middle). In this model the client had three separate endpoints available. In a normal operation, the client would call first the descriptor API (on the bottom of the server) to check if the computation he is seeking is implemented on the server. Secondly, the user would upload an image or video hitting the audiovisual API. Finally, the user would call the computation API asking for a descriptor to be computed on the image with a given unique id. The URL would look like the following for an image descriptor:

http://aesteassist-server.com/image/275/?descriptor=colorfulnessImage

For video it would only be necessary to change the *image* field for *video*:

http://aesteassist-server.com/video/275/?descriptor=colorfulnessVideo

4.3.3. Final Architecture Version

The last version proposed had several production-based improvements. An administrator console was needed in order to maintain and control the server from any browser window given proper permissions. Also, the audiovisual model was simplified. The video entity was modified in the model to become super class of the image. In other words, image was nothing but a

subset of a video. The frame resource was then removed from the model. Authentication and permissions was the final addition to ensure a production ready application.

This final architecture deeply explained in chapter 4 and an overview can be seen in figure 7 (bottom). Also, the final model is thoroughly discussed in chapter 4.2.

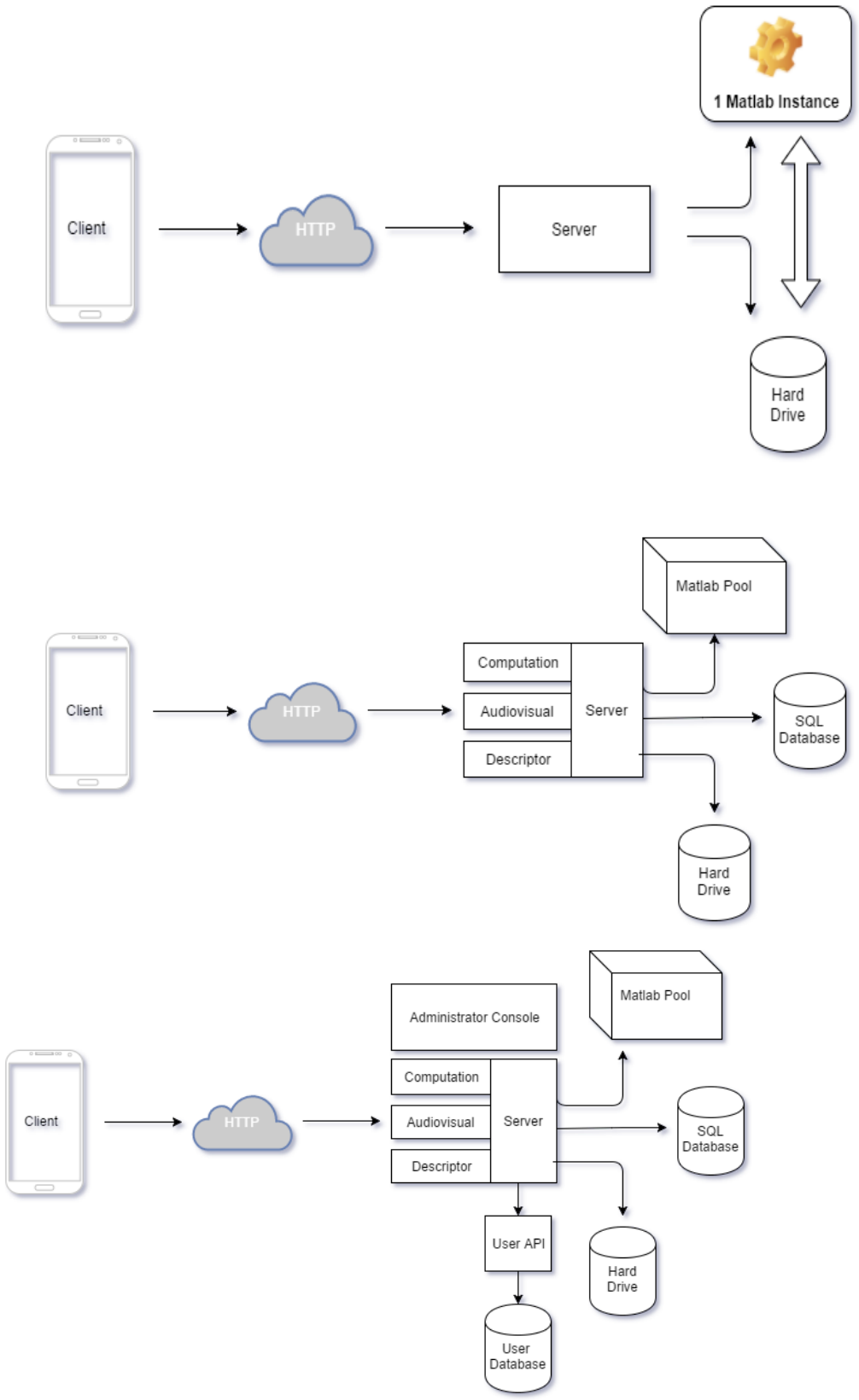


Figure 7 – Figure showing the three consecutive versions of the architecture. The top figure represent the first version. The figure in the middle is the second version. The figure at the bottom shows the final architecture.

4.4. Testing Phase

After the first final version of the project was finished the plan was to spend some roadmap time on reiteration. In other words, testing edge cases where the software was prone to fail and rebuild solutions to those cases. Members of the AP research team, main users of the AffectivePixels architecture, ve also helped me identify some unexpected functionality within the system. In a typical software development pipeline this could be done by a QA engineer (Quality Assurance engineer). However, the major part of the server side testing was automated by including unit, integration and load tests.

The client testing was mostly planned to be at the same time of app development. The benefit of Ionic is the use of a live reload server running on the computer that watches for changes on the source code and sends them directly to the application running on mobile phone. This type of development is very common when using web technologies to develop mobile apps.

4.5. System Requirements

Following the Standard-830 as set by the IEEE [20] the requirements should meet a specific template defined by:

- Unique ID: Each requirement has to include a unique identifier
    - Value: (CODYY) COD is a letter code relevant to the precise requirement we are defining and YY is a number provided in sequence order
- Name: The name should be somehow representing the requirement and include who does the action.
- Dependency: Determines which implementation it depends on.
- Description: Paragraph review that reveals all information needed about the requirement to be programmed

It is important to note that necessity and priority field present in the 830 standard have not been added to these tables for simplification purposes.

4.5.1. Software

These set of requirements refer specifically to the software needed to run the AffectivePixels server.

Table 4 – Python Installation system requirement

Unique ID	SYS01
Name	Python Installation
Dependency	
Description	Install Python through Anaconda Python Distribution

Table 5 – MySQL Installation system requirement

Unique ID	SYS02
Name	MySQL Installation
Dependency	
Description	Install MySQL on the system to be used as the persistence scheme

Table 6 – Matlab Installation system requirement

Unique ID	SYS03
Name	Matlab Installation
Dependency	
Description	Install Matlab with a license greater that R2014b

4.5.2. Local Dependencies

These refer to the packages that are used in the AffectivePixels server and should be installed through the Anaconda Package Manager or otherwise through pip. This requirements have been simplified by wrapping them on a single python file so that the only requirement is to run that file.

Table 7 – Dependencies local requirement

Unique ID	LOC01
Name	Affective Pixels Dependencies
Dependency	
Description	Run the install_dependencies.py file with contains all *pip* and *conda* calls to install needed dependencies

4.5.3. Hardware and Operative System

The AffectivePixels server has been tested on Windows and Ubuntu machines. Therefore, even though it should work on any platform with even less capabilities, the requirements for a correct functionality have to meet at least the tested computers specifications.

Table 8 – Windows Hardware requirement

Unique ID	HAR01
Name	Windows Distribution
Dependency	
Description	Windows 8.1 Intel core i3 RAM 4Gb 2 cores

Table 9 – Ubuntu Hardware requirement

Unique ID	HAR02
Name	Ubuntu Distribution
Dependency	
Description	Ubuntu 14.04 LTS Intel core i5 RAM 6Gb 2 cores

4.6. Functional Requirements

These requirements are related to the services that the AffectivePixels client/server architecture should offer. It is divided in Web Server requirements and Client requirements.

4.6.1. Web Server Requirements

In order to have a web server that provide all the functionality needed for image and video analysis we have to meet these specific web server application requirements.

Table 10 – User Web Server requirement

Unique ID	SER01
Name	User endpoint
Dependency	
Description	An HTTP endpoint to create and authenticate users with a Token

Table 11 – Descriptor Web Server requirement

Unique ID	SER02
Name	Descriptor endpoint
Dependency	SER01
Description	An HTTP endpoint to Create, Retrieve, Update and Delete a descriptor. It should also be able to set auto dependencies on other descriptors. Only Administrators should be able to use these endpoint except for retrieve which should be available to any user (list descriptors available)

Table 12 – Audiovisual Web Server requirement

Unique ID	SER03
Name	Audiovisual endpoint
Dependency	SER01
Description	Upload and Download images and videos. These endpoints should be restricted to unauthorized users.

Table 13 – Compute Web Server requirement

Unique ID	SER04
Name	Compute endpoint
Dependency	SER01, SER02,SER03
Description	An HTTP endpoint to compute a descriptor on an audiovisual resource. It should construct and resolve a dependency tree on the descriptor. These endpoints should be restricted to unauthorized users.

Table 14 – Admin Web Server requirement

Unique ID	SER05
Name	Administrator console
Dependency	SER01, SER02, SER03
Description	An Administrator console from which users with administrator access can control the server, add new descriptors and dependencies, eliminate users and delete audiovisuals.

4.6.2. Client Requirements

These set of requirements define what kind of functionality a mobile application should offer to the user.

Table 15 – Cross Platform Client requirement

Unique ID	CLI01
Name	Cross Platform
Dependency	
Description	The client application should be developed using cross platform techniques to program only once and deliver to multiple OS.

Table 16 – Get or take image Client requirement

Unique ID	CLI02
Name	Get an Image or Take it
Dependency	
Description	The client application should be able to retrieve an image from the filesystem or be able to take a new one using the camera

Table 17 – Get or take video Client requirement

Unique ID	CLI03
Name	Get or take video
Dependency	
Description	The client application should be able to retrieve a video from the filesystem or be able to take a new one using the videocamera

Table 18 – Http requests Client requirement

Unique ID	CLI04
Name	Http requests
Dependency	
Description	The client application should be able to make http requests to upload an audiovisual, download an audiovisual, ask for a computation and retrieve the result

Table 19 – Display results Client requirement

Unique ID	CLI05
Name	Display results
Dependency	
Description	The client application should be able to display the results in some visual way to show the value of the descriptor

5. Architecture Description

This chapter thoroughly describes the different parts and functionalities of the AffectivePixels image and video analysis Client / Server System. After going through a general overview of the system this chapter will be divided in two parts, server side architecture and the client side architecture. First, on the server explanation, the main access points will be described as well as the general data model that was used. Then, on the client side, a walkthrough of the different views will be discussed explaining why where they developed and what their main functions are.

5.1. Overview

The final architecture design that was mentioned in section 3.3.3 is now listed in detail. Taking a look at figure 8 it can be seen that there are five main entry points to the server:

- Administrator console: serves as a maintenance and control page for users with administrator privileges to create and modify the descriptors that the server implements, user management tasks or audiovisual deletions.

- Computation API: this access point interface serves as the place to ask for a computation given the user has permissions on the image and the descriptor actually exists.

- Audiovisual API: is the one in charge of uploading and downloading images and videos. It also indexes them in the database following the Audiovisual Model described in section 4.2.

- Descriptor API: this access point handles the creation, modification and deletion of descriptors. By adding a new descriptor, the administrator is basically adding functionality to the server. It is important to note that the Matlab .m code has to be added to the server too.

- User API: Authentication and permissions are a key part of any software system. Therefore, this endpoint handles user tokens that expire in time and completely identify a user and corresponding permission.

The server also has other key parts that are not Django related. This includes the Matlab pool of instances, the MySQL database and the hard drive memory.

- Matlab Pool: A set of non-desktop instances of Matlab capable of running Matlab functions and returning a response. The amount of instances is dynamic and configured by the administrator.

- MySQL: A SQL database where all the images, videos and descriptors are indexed for future retrieval. SQL queries are not directly made as Django models provide with ORM (Object relational mapping) functionality.

- Hard drive memory: Common I/O memory to store a large amount of files in a folder based structure.

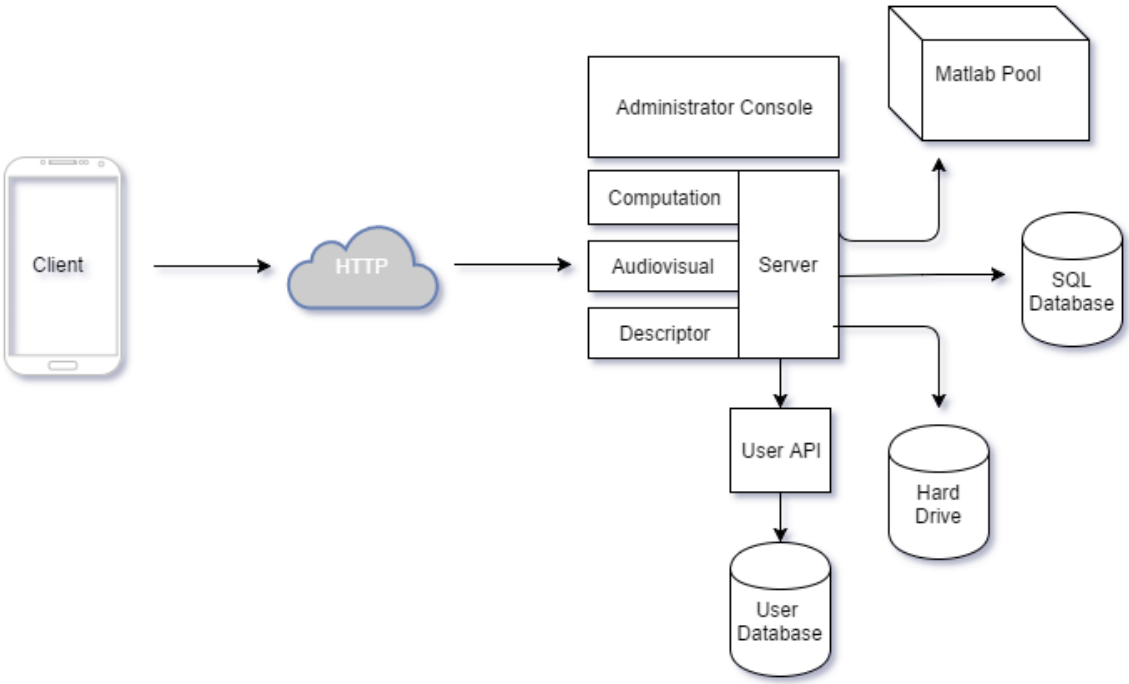


Figure 8 - Final Architecture flowchart design.

5.2. Web Server Description

The server is the key part of the AffectivePixels architecture. It provides the system with computation capability, image and video storage, result indexing and user authentication and permissions. Most of the work of the development period has been done in the web server as it holds the main business logic. This section will explain the different services the server offers as shown in figure 9.

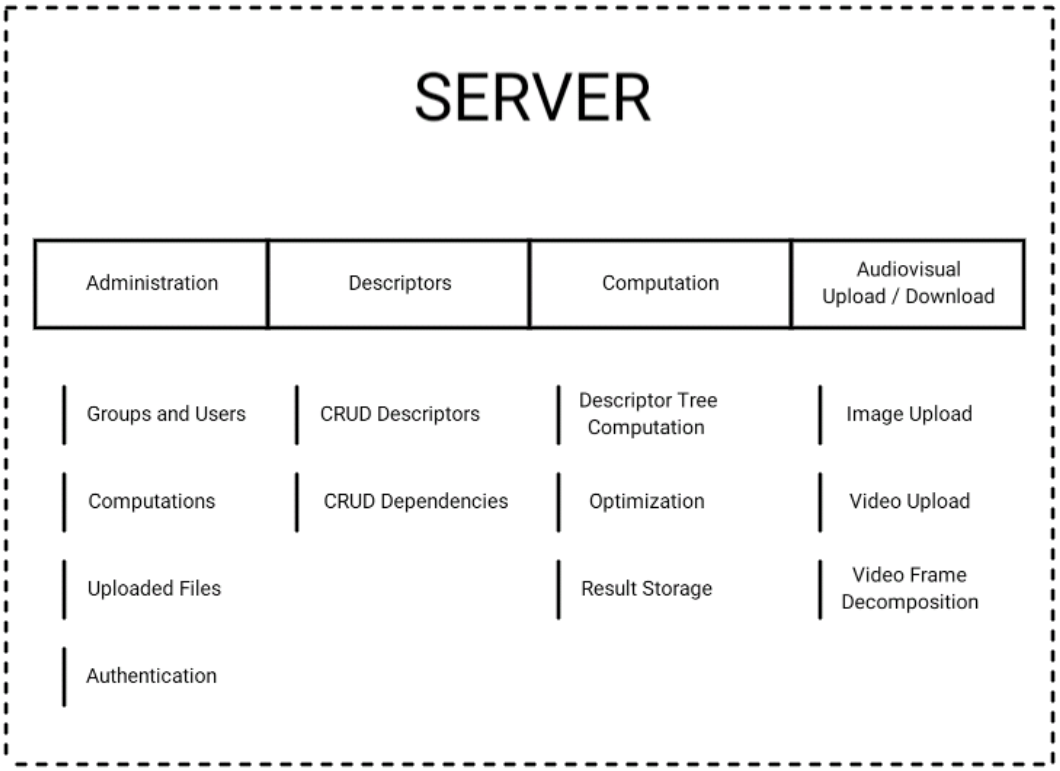


Figure 9- Web Server Endpoints. All functionality endpoints are shown except the User resource.

5.2.1. Audiovisual Model

In order to store images, videos, descriptors and the computed results of these resources and its dependencies it is crucial to have a very representative and as simple as possible data model. It is a final revision made from the audiovisual model proposed in the second architecture from section 3.3.2. This final version introduces a new perspective by removing the frame entity and considering it just another image. On top of the pyramid there will be a video resource which has a set of images as dependencies. The case of single image uploading fits the model by considering it has a null video parent. This behavior is represented in figure 10.

The Video Table contains the properties needed for the complete identification of a given video resource. The Image Table is very similar. The only difference is a foreign key field to the Video Table since images may belong to a video (if they are frames).

A primary keys relation table named Computed Descriptors has been used to store computation results for images and videos. This table contains two nullable foreign key fields to the Video Table and Image Table in order to fit both cases. The result is stored as a JSON string that can be easily parsed by the server. The reason for this is to solve the heterogeneity on

the descriptor result. A given descriptor may return a matrix, a scalar or, for instance, a Matlab cell array. Storing a JSON string homogenizes the result storage. This represents a great advantage for the server permitting different technologies to be introduced in the future with the only requirement of giving a JSON object as output. Therefore, one of the main objectives listed in section 3.3 is now accomplished.

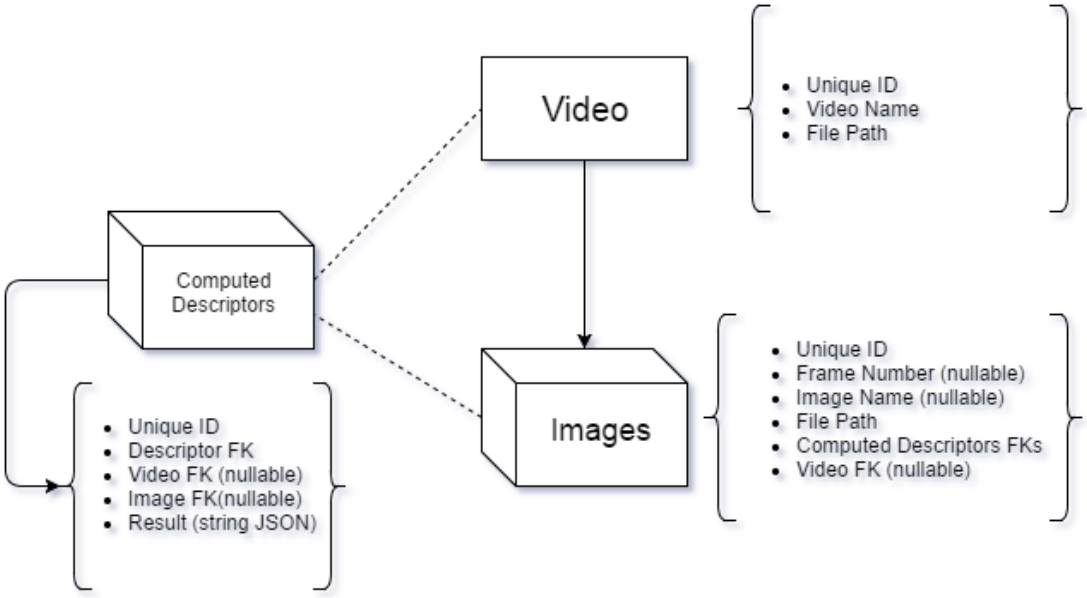


Figure 10 - Final Audiovisual Model that includes a video as a parent resource which has images as children resources. They both have a computed descriptors liking table to store already performed descriptor computations.

5.2.2. Audiovisual API

The audiovisual API is used to upload and download images and videos. Also to index those images in the SQL database.

For Image and video uploads multipart requests have been implemented. Django provides very good support on this [36]. In order to save the image and video files in the model Django File Fields have been used with a custom formatting function to customize the file path.

- For images:

  %SERVER_FOLDER%/<user_id>/images/<image_id>.<extension>

- For videos:
  %SERVE_FOLDER%/<user_id>/videos/<video_id>.<extension>

- For decomposed frames:
  %SERVE_FOLDER%/<user_id>/images/<video_id>/<frame_number>.jpeg

Video uploads are decomposed into frames from the very first moment they arrive to the server. The current Affective Pixels research, almost in every case, requires videos to be decomposed into frames. Therefore, it has been decided to always make the decomposition as an optimization task. A multiplatform library has been used for video decomposition: *FFmpeg* [7]. This tool uses lower implementations which provide "added value and effectiveness" when working with a "translation level" [37] in a cloud computing environment.

5.2.3. Descriptor Resource

In the Affective Pixels Architecture computations are organized around descriptors. A descriptor is a model that relates a scientific concept to a Matlab function that can compute a value measuring that concept. The different properties of the descriptor are represented in figure 11.

The Descriptor API is entirely dedicated to creating, modifying and deleting descriptors. It has restricted access to any user without administrator permissions. By using the Generic Mixins [38] from Django Rest Framework, which provide common functionality, CRUD (create, retrieve, update and delete) operations were quickly developed. These operations are directed directly towards the MySQL database.

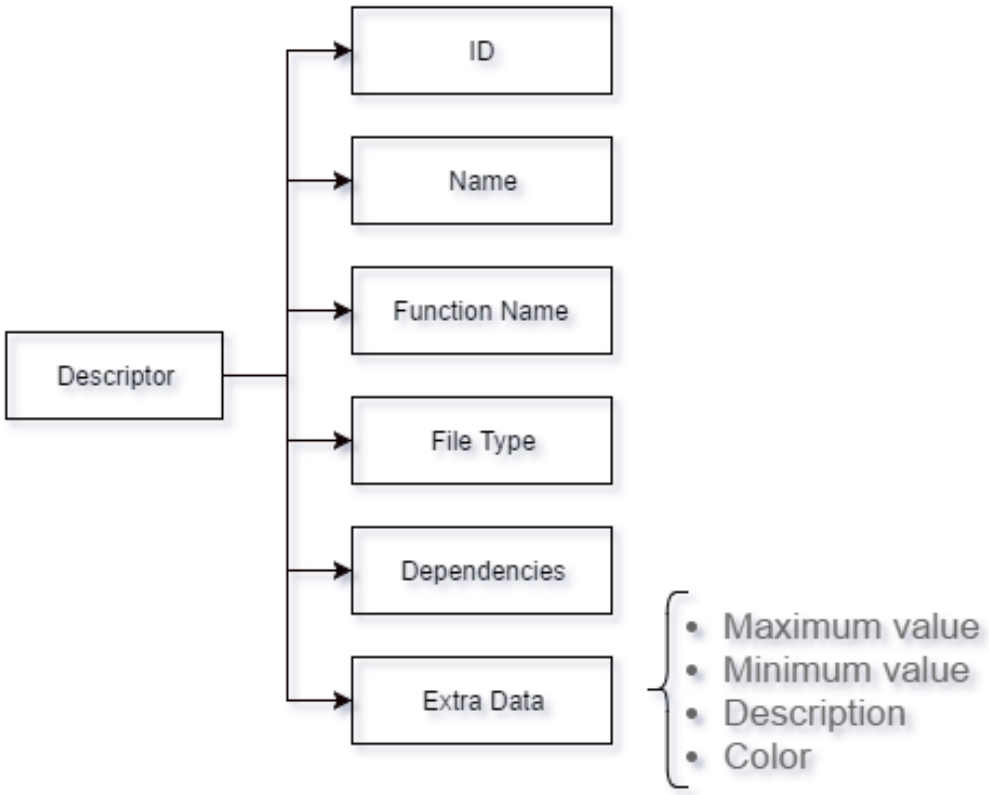


Figure 11 - Descriptor model.

A descriptor resource has:

- Unique Identifier: Integer number that uniquely identifies the descriptor.

- Name: Word that completely describes the descriptor. It is alphanumeric and unique. It is the property by which the user will ask for a computation. In other words, this is the property present in the http request query set to identify a descriptor.

- Function Name: String that represents the name of the Matlab function related to this descriptor.

- File Type: Either Image or Video. It refers to the descriptor target file type.

- Dependencies: Set of other descriptors that this one depends on. It may be null. If there are dependencies they will be computed first

when the computation tree is resolved. This is thoroughly explained in section 4.2.4.

- Extra Data: Complementary information that can be used by the client to make a graphical representation of the numerical result.

5.2.4. Computation Resource

This section is divided into three subsections that together make up the Computation Resource. Each one of them is explained in detail in their specific subsection.

It is important to note that this resource represents the key functionality of the Affective Pixels Architecture. It should provide a fast and invisible cloud calculation of the descriptors supporting many parallel connections. These implementations were designed to support any descriptor dependency case and to elicit the smallest footprint possible on the server computation memory.

5.2.4.1. Matlab Pool

A Matlab pool has been implemented leaning on the Pymatbridge [34] library. It can be observed in figure 12 how the server derives incoming computations to the less busy Matlab instance at any given moment. In order to do so the logic has been divided into three modules:

- Instance Finder Module: A piece of code that checks through an in-memory counter which is the less busy instance. Once the freest has been detected from the Instance Calculation Counter array the Finder Module adds one to that counter.

- Compute Module: This part is in charge of the communication to the instances. A ZMQ socket [39] (TCP networking socket with extra functionality) is opened by the Pymatbridge library which serves as the communication tunnel. Input arguments are sent to the Matlab function and once it finishes the result is received back in the Python code.

- Free Instance: This module controls the release of an instance calculation. Its task is to subtract one from the instance counter at its specific position.

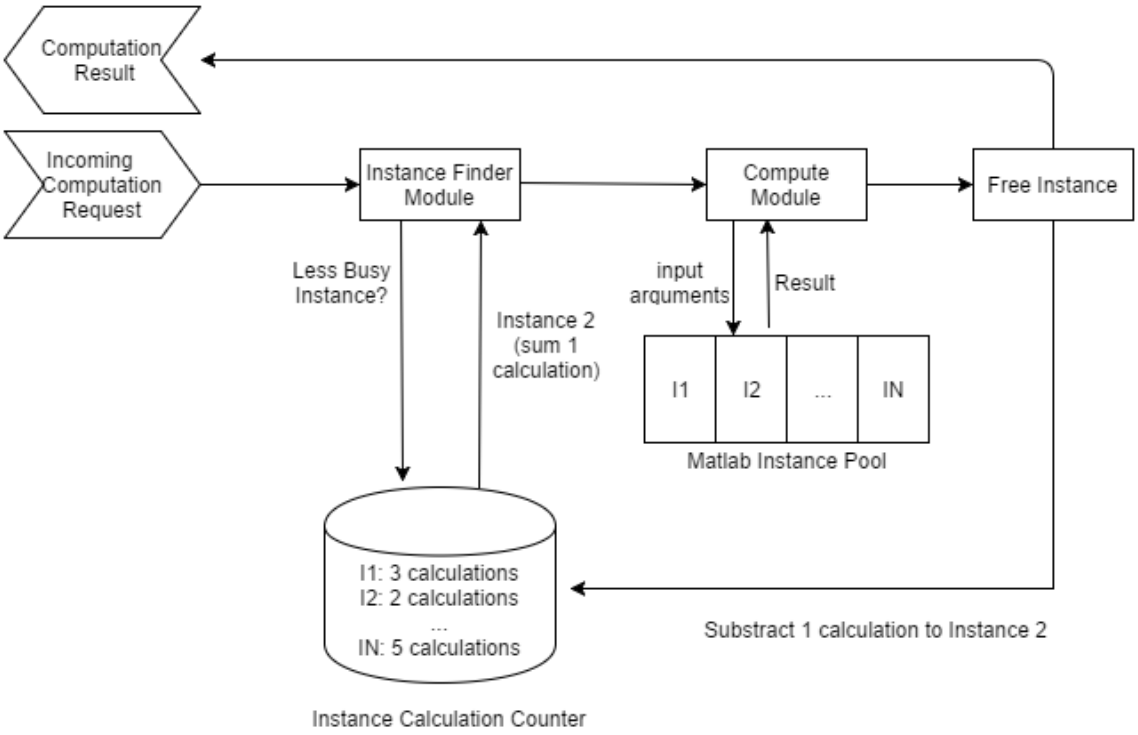


Figure 12 - Matlab Computation Pool. A complete representation of the system developed to forward computations to Matlab instances.

The way the Matlab Pool includes dependency arguments to descriptor computations is represented in figure C.1. All the descriptor implementations should be programmed following the Annex III Matlab Programming Guideline. Every descriptor function accepts only one argument, usually called *args*, which is a struct containing all the dependencies. The dependency names are the keys of the struct and the values of the struct are the computed dependency values as explained in detail in Annex III for image and video cases.

5.2.4.2. Image Dependency Tree

A descriptor represents a measurable characteristic of an image that can be computed through an algorithm. That descriptor may depend as well on other descriptors. One good example could be a general *image quality descriptor* that tries to assess image quality by fitting to a linear model the result of other four descriptors: colorfulness, intensity, rule of thirds and entropy. In this specific case, the other four descriptors should be calculated beforehand in order to be able to compute the *image quality descriptor.* This concept is presented as the dependency tree which has to be resolved in a general way for a correct computation functionality.

In this section only Image Descriptors are analyzed. They are a simpler version of descriptors which only are valid for images and therefore its tree resolving implementation is simpler. Figure 13 represents the general Image Dependency Management algorithm developed in this project. The very first part of the algorithm is a validation step that assesses the belonging of an image resource to a given user. After that passes, the actual dependency management begins:

- Step 1: Check if the descriptor has been already computed and if it's present in the database. If it has, retrieve it and return the result.

- Step 2: If the descriptor has not been computed already the algorithm has to check for dependencies. If it doesn't have any, it will proceed to call the Matlab Pool for a direct computation request.

- Step 3: If there are unresolved dependencies the method calls itself again in a recursive manner for every dependency found using a for-loop.

- Step 4: Every time the Matlab Pool returns a valid result it is stored in the purple Dependency Dictionary. The dependencies are taken from this in-memory databank to provide the arguments needed for a Matlab Pool Computation of a descriptor with dependencies.

- Step 5: At this point all the dependencies and sub dependencies have been resolved and stored in the databank. The final computation is derived to the Matlab Pool which imports as arguments all the dependencies from the Dependency Dictionary. The descriptor is finally computed.

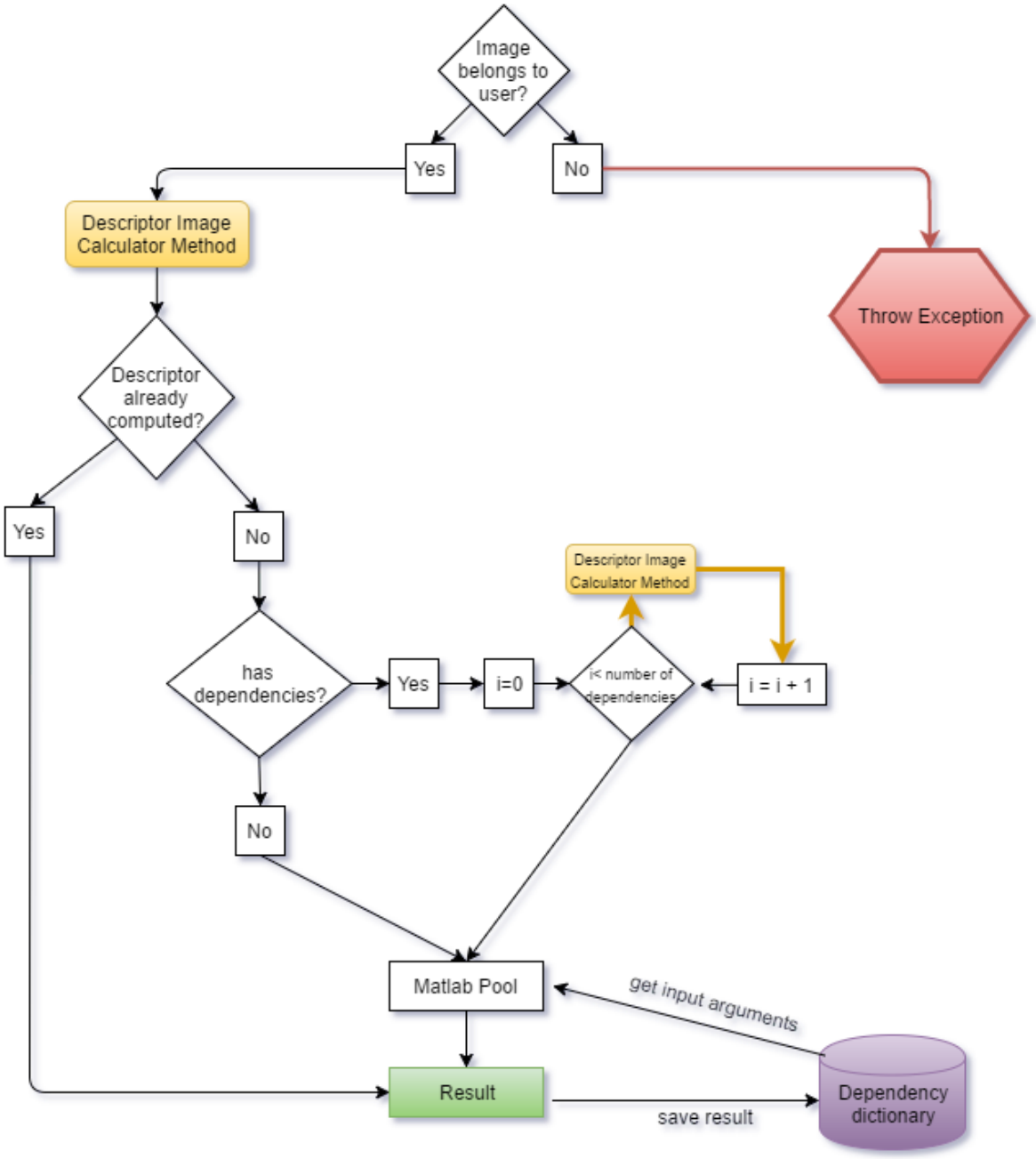


Figure 13 - Image Dependency Tree. A decision tree that resolves descriptors dependencies before computing the final descriptor.

5.2.4.3. Video Dependency Tree

The server has to handle Video Descriptor computations as well. These require an algorithm that generalizes the previous image-only case. It is important to note that videos are decomposed into frames (images with a video foreign key in the audiovisual model) and therefore any image dependency of a video means that an image descriptor has to be calculated on every single frame that the video has.

The implementation shown in figure 14 looks somehow similar to the image-only case but includes the Complete Dependency Management Module. This code region will loop through all the dependencies checking for their file type. In the case of a video dependency the module will iteratively call the Video Calculator Method (shown in orange) and store the result in the Dependency Dictionary (shown in purple). In the case of an image descriptor, another loop will fire the Image Calculator Method from previous section 4.2.4.2 for every frame that belongs to the video. The module will then store frame computation results as a key/value pair object where the key is the descriptor name and the value is an array where each position relates to the frame at that position.

After the Complete Dependency Management Module is explained the step-by-step process is simply the following:

- Step 1: Check if the video descriptor has been already computed and if it is present in the database. If it has been computed, retrieve the result and return it.

- Step 2: If the descriptor has not been computed already the algorithm has to check for dependencies. If it doesn't have any it will proceed to call the Matlab Pool for a computation request.

- Step 3: If there unresolved dependencies the method calls the new Complete Dependency Management Module.

- Step 4: Once the module has resolved every dependency the algorithm will call one last time Matlab Pool to compute the video descriptor using all the dependencies available at the Dependency Dictionary (shown in purple).

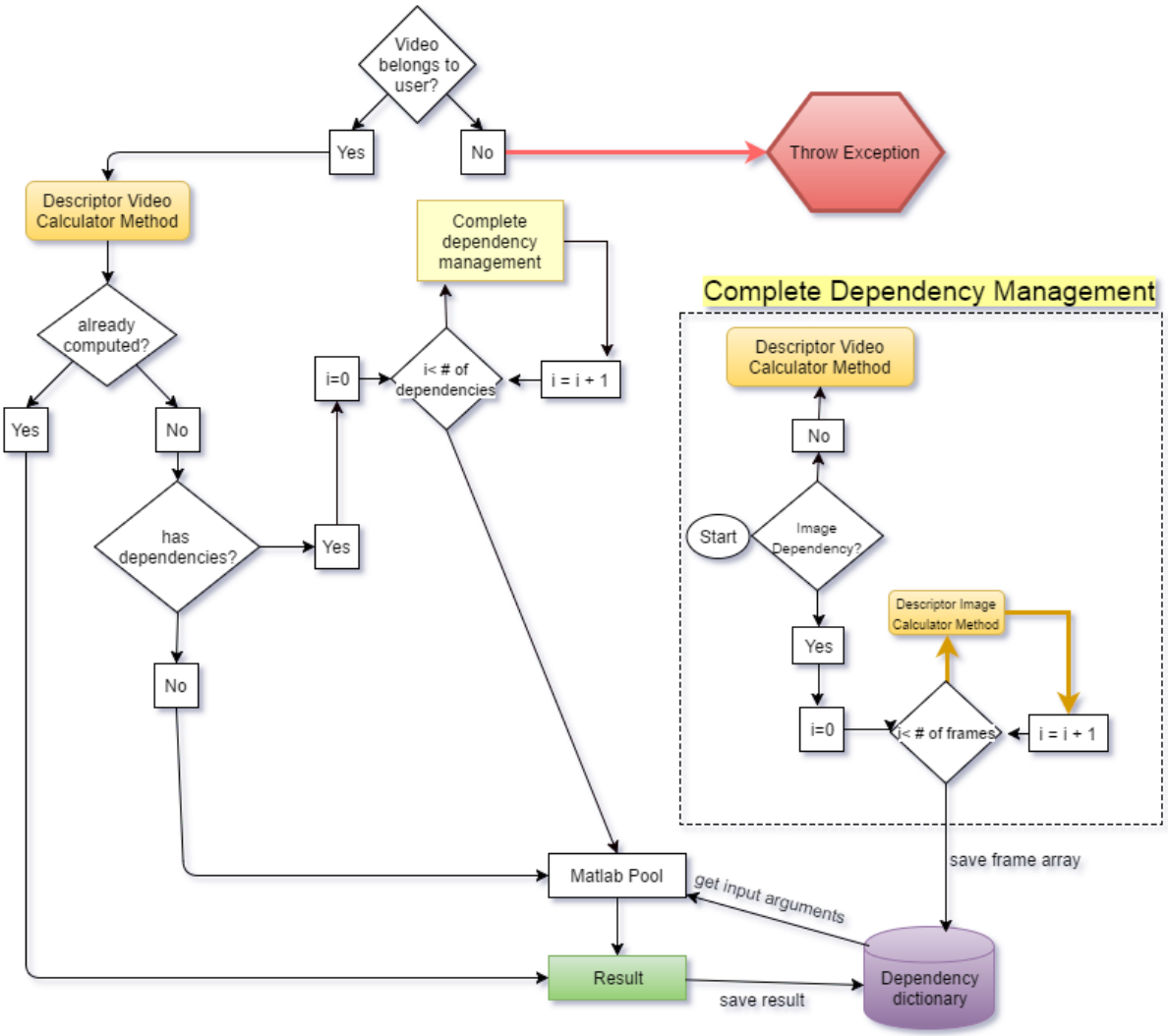


Figure 14 - Video Dependency Tree. An improvement of the image dependency tree that resolves descriptor dependencies before computing the final descriptor.

5.2.5. User Resource

Authentication and permissions are a crucial step to deliver a production-ready system. The administrator of the server has the ability to create and delete user accounts. This is the way to restrict the access to the online AffectivePixels server to only trusted clients. Users of the Architecture will only be able to see and work with their own images and videos and never with other user's resources.

User authentication leans on the Django User Model [40] which brings SHA256 encryption for sensitive data like passwords. This prevents a hacker could have breached the system to retrieve valuable dangerous information. Several authors of applied cryptography books state that predecessors of SHA256 had "various weaknesses" but the new encryption does "not cause any threat to the security" according to their experiments on hash functions [41].

The chosen authentication method has been *Json Web Tokens* [42] that implement a simple expiring token validation technique represented in figure 15. The very first request the client sends is to the User API:

https://affectivepixels-server.com/user/token/

The payload of that request includes the username and password of the user. It is important to note that the method sends a JSON object in the request body that includes sensitive information. Therefore, for security reasons, it is essential to use the HTTPS transport method to ensure security in a production environment. If the username and password are validated by the server, a JWT is automatically issued and sent back on the HTTPS Response. The token has an expiration date of 8 days set by the administrator configurations. For the future requests, the client has to include that token in the header Authorization key. This will permit the server completely authenticate the user and apply any corresponding access permissions.

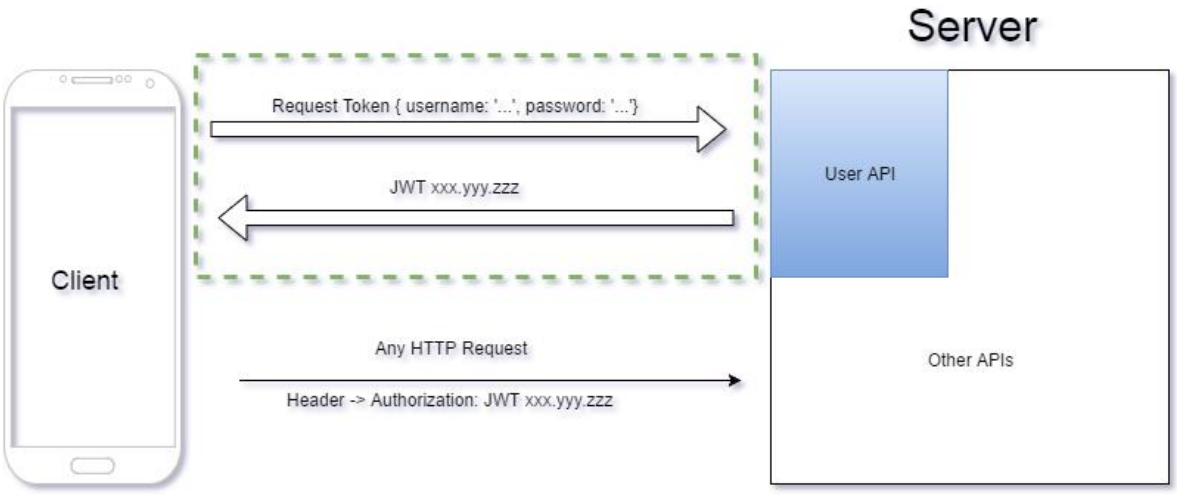


Figure 15 - Authentication API. Surrounded in green it can be seen the main step where a JWY is returned to authenticate a user.

The JWT token has a very specific form. It is a JSON object encrypted by a SHA256 algorithm that includes the secret encryption password as a property. When the server decodes an incoming JWT token it first validates the value of the secret key. If that value matches the existing one in the server, the token can be considered valid and username is extracted. In the case that the secret key is not found in the decoded token or it doesn't match the existing key on the server,

the JWT is rejected. Figure 16 shows the two states of a token, encoded and decoded.

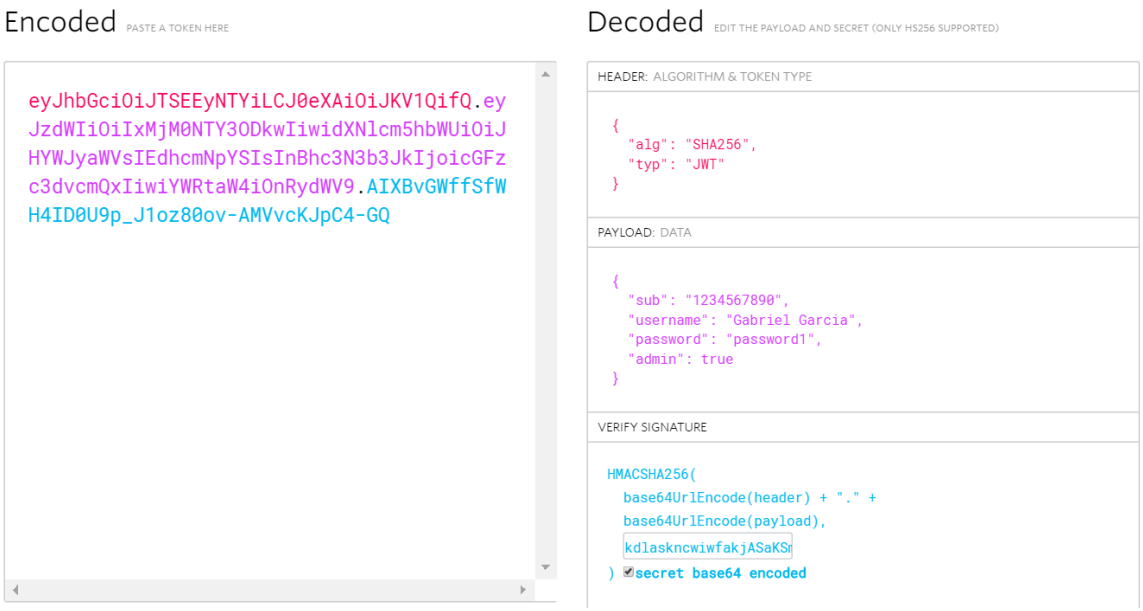


Figure 16 - JWT encoded and decoded.

5.2.6. Administrator Console

The server should be easily controlled by a manager without the necessity of editing a file. This service is provided by a console that can be accessed through the URL where the server lives:

http://affectivepixels-server.com/admin/

This user interface lets the administrator perform different tasks:

- Descriptors: Create new descriptors, edit existing ones and delete unneeded ones.
- Audiovisuals: It also has built-in functionality to query and modify images and videos in a table-like manner.
- User management: It's the place to add new users, edit permissions of them and delete old ones.

All of these tools are listed in figure 17. There, a welcome page is presented where the administrator can navigate through the different control pages and check the recent activity of the server.

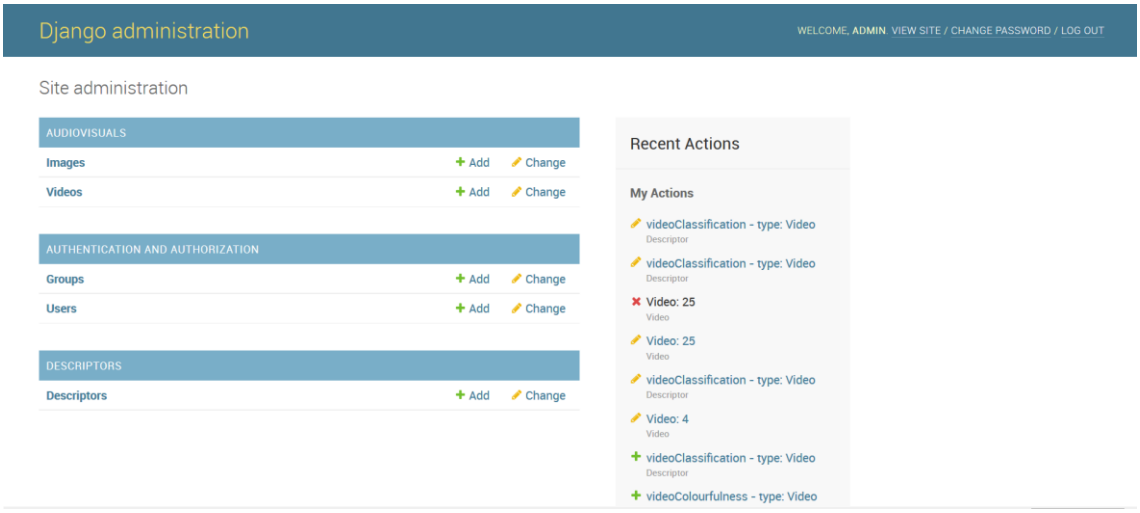


Figure 17 - Administrator Console. Welcome Page.

The first functionality of the console can be opened by selecting the Descriptors link. Once clicked, a new page appears (figure 18) where all the descriptors are presented in an organized table. A selection filter column to the right of the page has been programmed to let the administrator query for a specific descriptor file type, either image, video or any type.

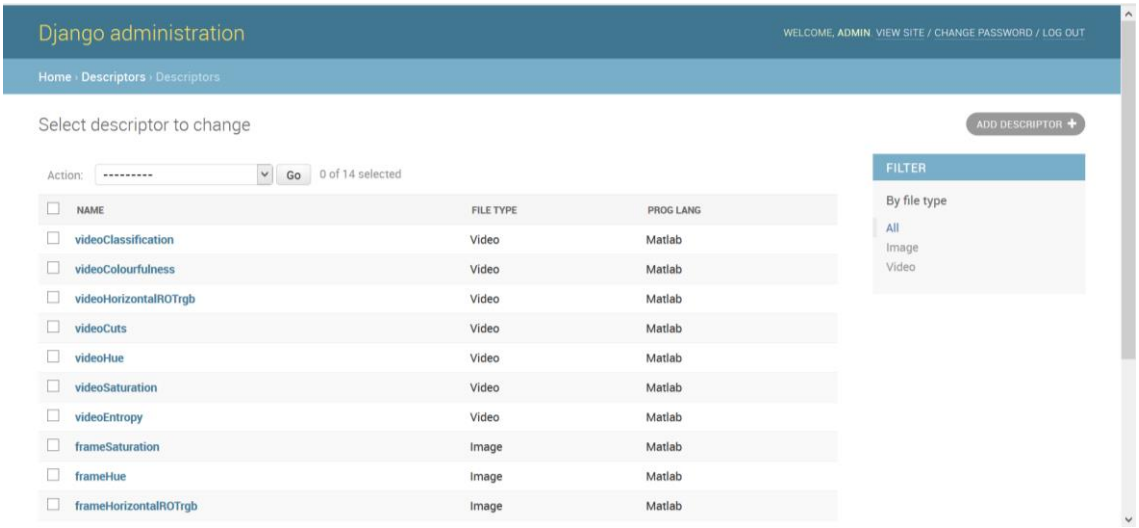


Figure 18 - Administrator Console. Descriptors Page where the administrator can add or select descriptors and query them by their file type.

From the descriptor detail page the administrator of the server can create new descriptors or edit existing ones. One good example is shown in figure 19 where the editing process is represented. In this specific case, the descriptor named *videoIntensity* has been selected. There are several fields to introduce and edit the various properties inherent to a descriptor as explained in the Descriptor Model in section 4.2.3.

It is important to note that dependencies, which are part of the descriptor model, are introduced here. If they have already been computed, like is the case of figure 19, the stored values are also shown. These are JSON string representations of the result as explained in section 4.2.1 to homogenize the heterogeneity of the computation results (matrix, vector, scalars...).

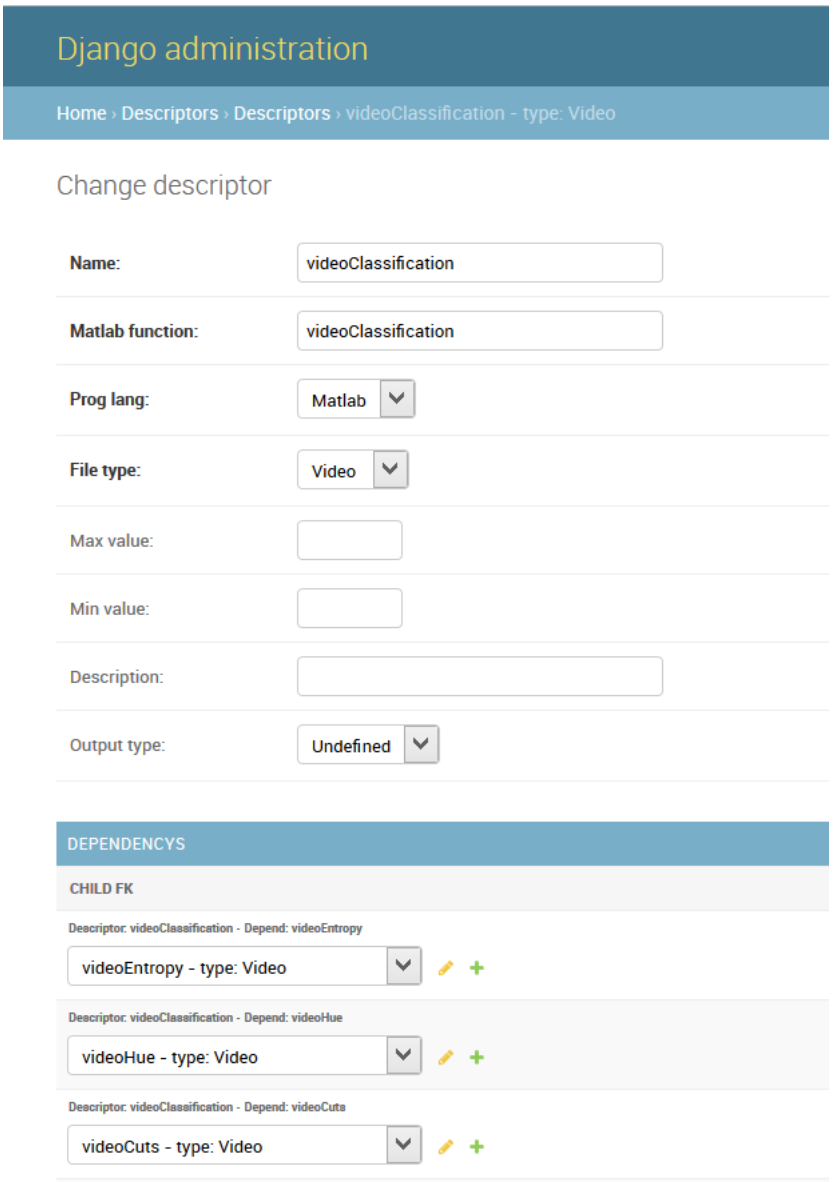


Figure 19 - Administrator Console. Descriptor Detail where a descriptor information can be modified and the dependencies edited.

The second functionality that can be accomplished with the administrator console is audiovisual management (figure 20). This section of the console presents a table where all the audiovisuals are organized. There is a query column to the right, as in the descriptor menu, to be able to find an audiovisual by video id and by owner username.

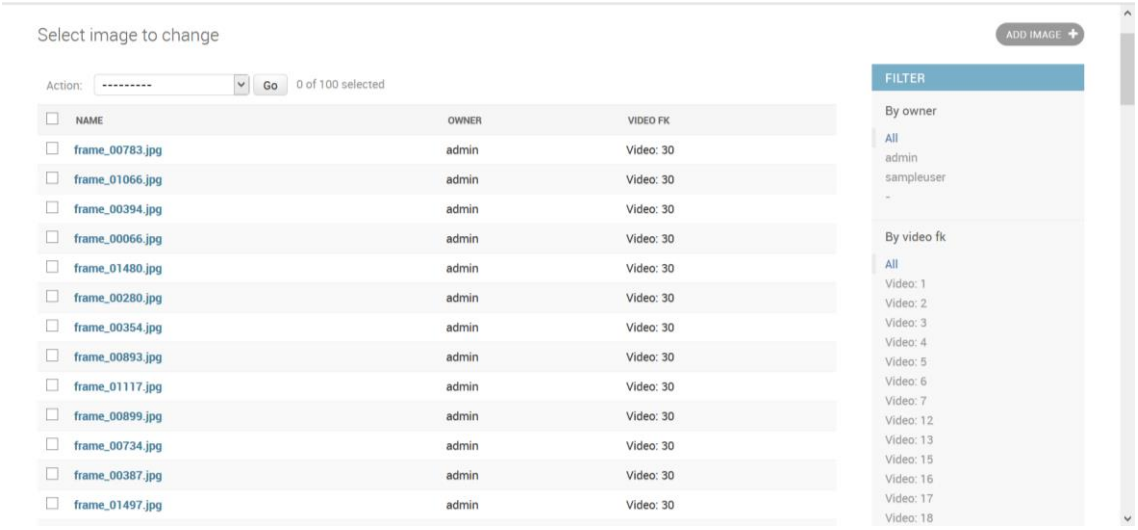


Figure 20 – Administrator Console. Images Page where the administrator can view a list of all images and query by video id or owner.

By selecting a specific audiovisual, the administrator is able to edit its properties of the video or image in the way shown in figure 21. The different properties belonging to an audiovisual model are presented.

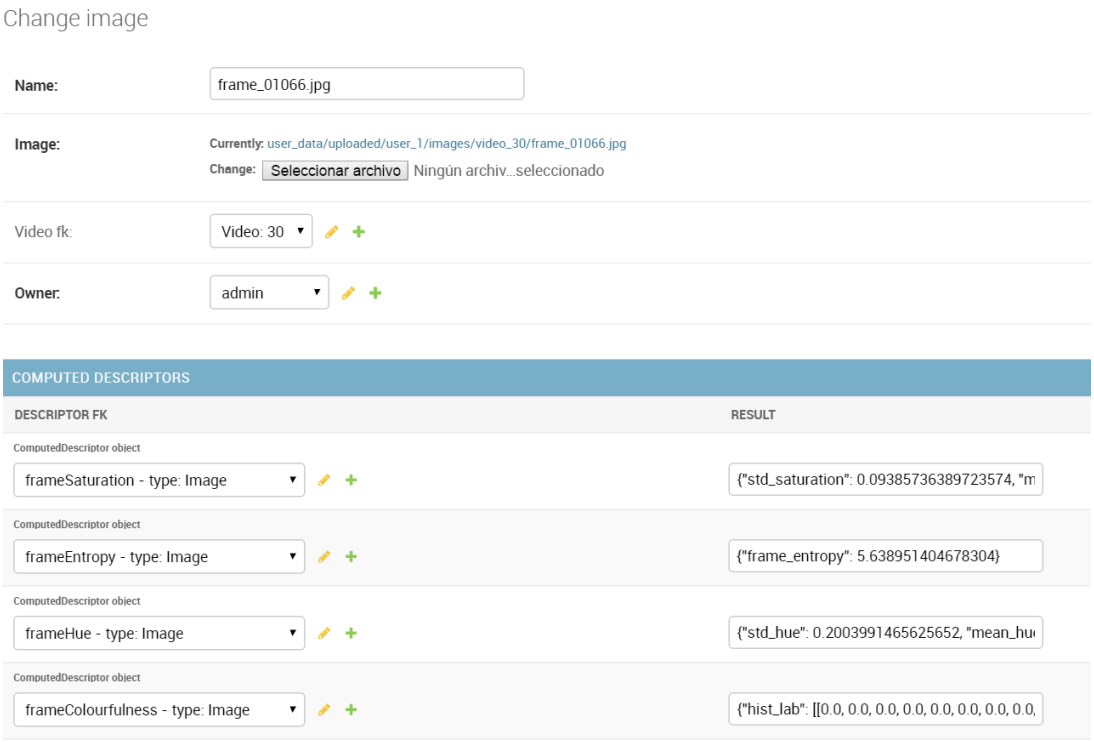


Figure 21 - Administrator Console. Audiovisual Detail Page where a specific image or video can be modified and computed dependencies (descriptors) are shown.

The third and last functionality of the console is to manage user accounts. From the user window the administrator is able to create a new user setting a username and password encrypted using SHA256. This prevents unauthorized use of the AffectivePixels Architecture. Figure 22 pictures how the window looks like.

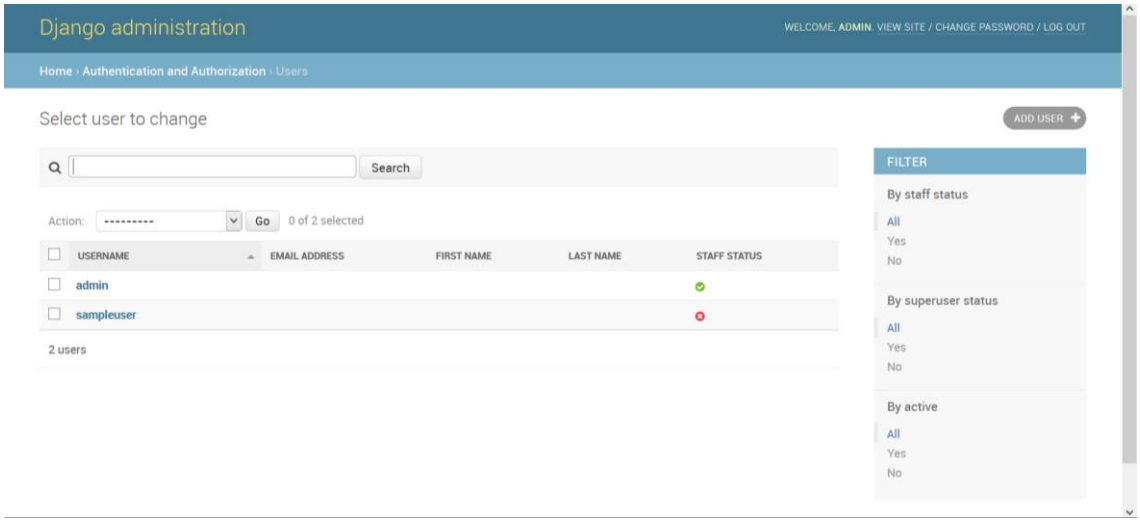


Figure 22 - Administrator Console. User Management Page where the administrator can create and modify users.

5.3. Client Description

This section describes the client application developed in this project to consume the Web Server endpoints and provide a user with well-presented interaction. The client application is required to be a multiplatform software that can be compiled from a single source code to both iOS and Android devices. It has been programed using the Ionic JavaScript framework and Cordova Plugins for native functionality, like accessing the camera or SQLite persistence.

5.3.1. AffectivePixels Descriptors

The members of the AffectivePixels research team have developed several descriptors that are used by the application. It is important to note that the client can work with any set of descriptors added in the administrator console, but the following are specific to the AffectivePixels study. Some of them are represented in the following tables:

Table 31 – AffectivePixels Image Descriptors

Descriptor	File Type	Description	Dependencies
Classification	Image	Classifies the perception of the image on bad, regular or good.	Colorfulness, Intensity, Entropy and Rule of Thirds.
Colorfulness	Image	Gives a value of the amount of color present in the image	None
Intensity	Image	Brightness of the image	None
Entropy	Image	Disorganization o the objects in the image	None
Rule of Thirds	Image	Measure of the rule of thirds application	None

Table 32 – AffectivePixels Video Descriptors

Descriptor	File Type	Description	Dependencies
Classification	Video	Classifies the perception of the image on bad, regular or good.	Saturation, Cuts, Entropy and Rule of Thirds.
Cuts	Video	Distribution of the different planes within a video	None
Saturation	Video	Distribution of the color saturation of the video and mean saturation	None
Entropy	Video	Distribution of the disorganization in the video and mean entropy.	None
Rule of Thirds	Video	Measure of the rule of thirds application to every frame and mean measure of the value.	None

5.3.2. User Experience

A fluid user navigation characterizes the best applications on the market. For this project, user experience has been designed on a simplistic basis where the least amount of words are written and a consistent interface is used across its various pages. A noticeable characteristic of the design is the large presence of icons instead of words on buttons. Also, cards are preferred over lists of items.

A tabular organization of pages has been implemented on this app since it increments simplicity of usage, moreover, many popular and successful applications like WhatsApp or Facebook have similar styles. It is remarkable how several studies on UX remark simplicity of interaction as a key property giving the application a more "professional and reassuring" appeal [43]. Ideally, common actions should be easily and rapidly carried out [44].

5.3.3. Login and Logout

The AffectivePixels architecture requires authorization to use its services. Therefore, the very first page has been selected to be a common user login page where the username and password can be introduced.

If the server verifies the credentials a JWT is sent back that is stored by the application in the SQLite database to be appended in the future to any other HTTP request as an authorization header.

This page is represented in figure 23. In the one in the right, it can be selected whether it is an existing user attempting to enter the system or if a new account should be created. For security purposes, there is no account creation functionality since this requires extra security like email verification or SMS verification which goes beyond the scope of this bachelor thesis. In the figure to the right it can be observed the two fields for user login.

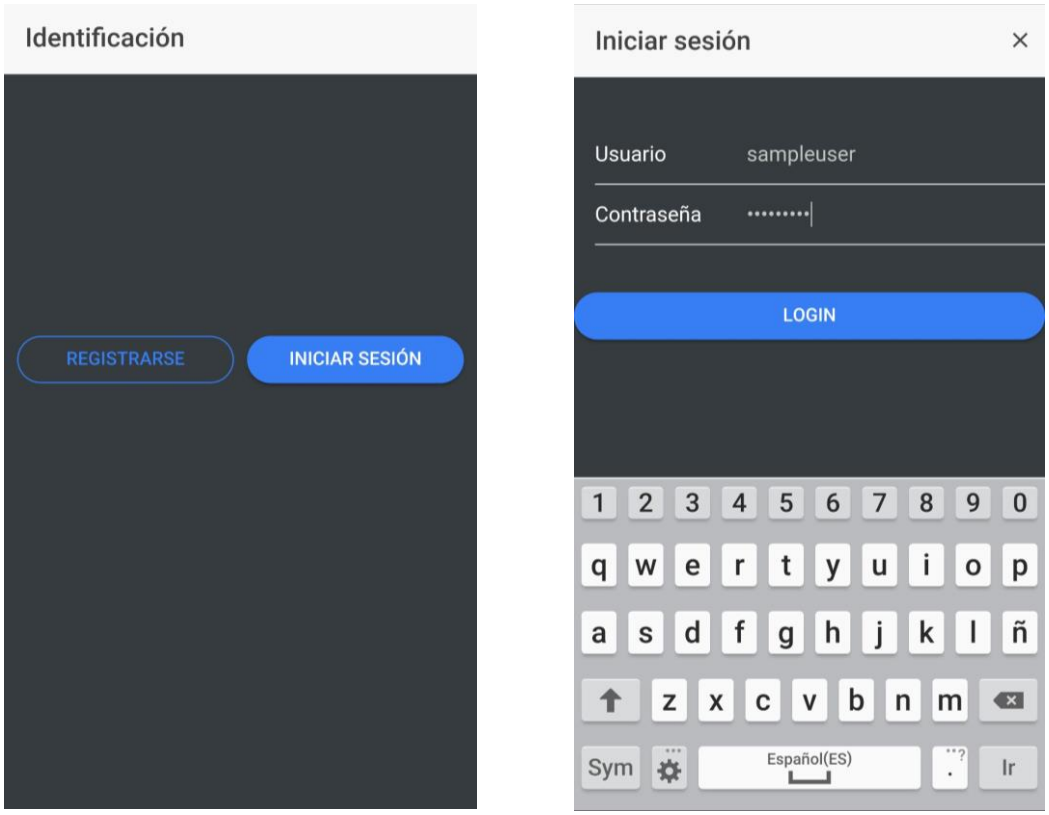


Figure 23 - Client Application. Identification page (on the left) and Login page (on the right).

5.3.4. Images

This page is intended to provide with image capture and analysis functionality. Figure 24 shows how this page looks like in an android device. The images listed there were used to test the rule of thirds descriptors. The different functionality achieved with this page includes:

- Take a picture: By clicking the footer blue button the native camera will be brought up for image capturing. Once the image is taken, the application:
    - Stores the image file inside the application folder. This folder is not accessible through the gallery of the device which has been chosen for privacy issues.
    - Indexes the file on the SQLite database

- Sends the image file to the server hitting the audiovisual image upload API.

- Query Pictures: All the taken pictures are shown as a scroll list where the user can select any of them. They are ordered by decreasing capture date.

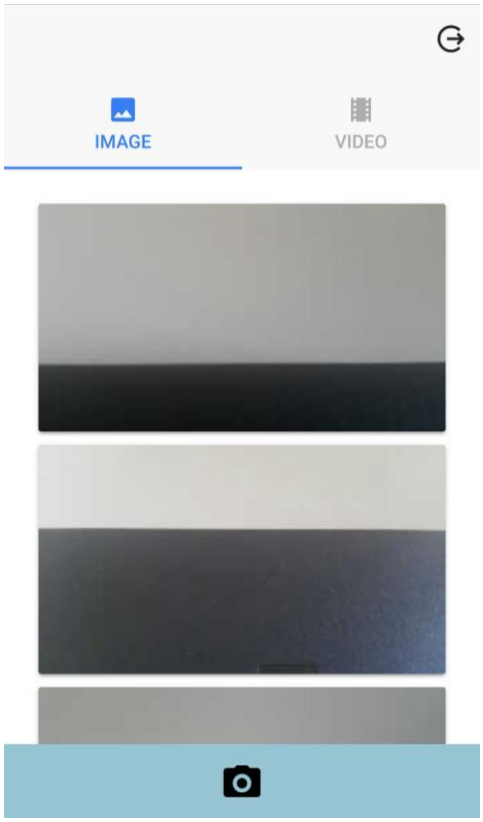


Figure 24 - Client Application. Image page (blue tab with image icon) that shows a list with all previously computed images and an image capture button at the bottom. These sample images are test samples for the rule of thirds descriptors.

From the list of images the user can select any of them. If for instance the second one is selected, a new page is opened (figure 4.3.2b to the left) where it is possible to request the computation of the different image descriptors to the server. By clicking the blue footer button with a calculator icon the list on the right side of figure 4.3.2b is presented where the available descriptors are shown. This list is populated with up-to-date data received by an http call to the descriptors endpoint requesting information about only those of image type:

http://affectivepixels-server.com/descriptor/info/image/

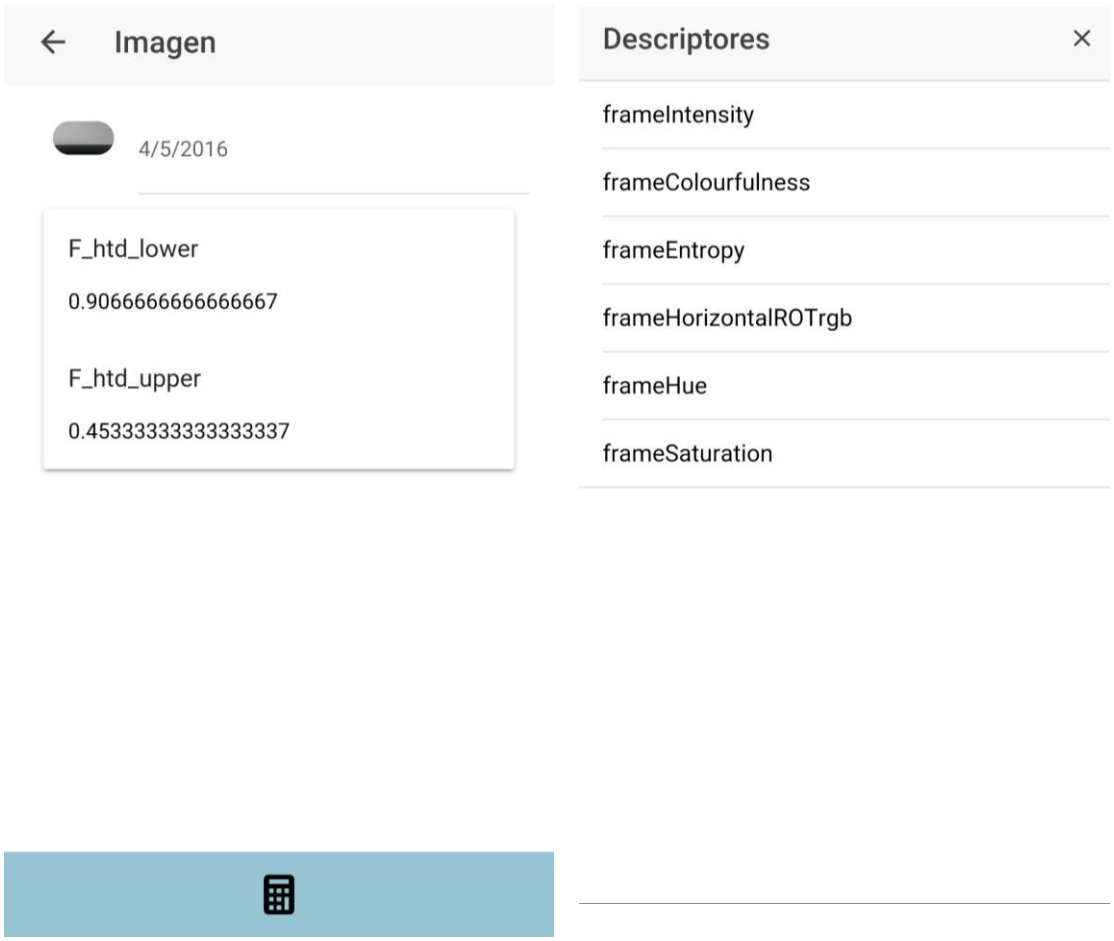


Figure 25 – Computations performed on the image and stored locally (left) and available descriptors list (right) that appear after clicking the lower blue button with a calculator icon

In figure 25, the rule of thirds has been previously computed. This descriptor attempts to measure if the image is either on the lower or upper third division of the image. As shown, the lower third has a higher normalized value (0.906) than the upper third (0.453). Therefore, the rule of thirds applies, and there majority of the contents of the image occur on the lower third. This concept exemplifies what a sample descriptor can achieve. The client can obtain information from an image by sending the image to the server and requesting a cloud computation on it, in this case, the rule of thirds descriptor.

Sample test images used in the AffectivePixels research for descriptor calculations are shown in the following figure 25b. The first two on top are the test images for the rule of thirds descriptors while the pair of images at the bottom are used as entropy references. As it can be seen on the pair on top, the image on the left has a clear rule of thirds application while the other on the right shows distinct values since rule of thirds is not present. On the other hand, the image on the left of the second pair shows a high

entropy value while the one on the right has a very homogeneous value, in other words, small entropy.

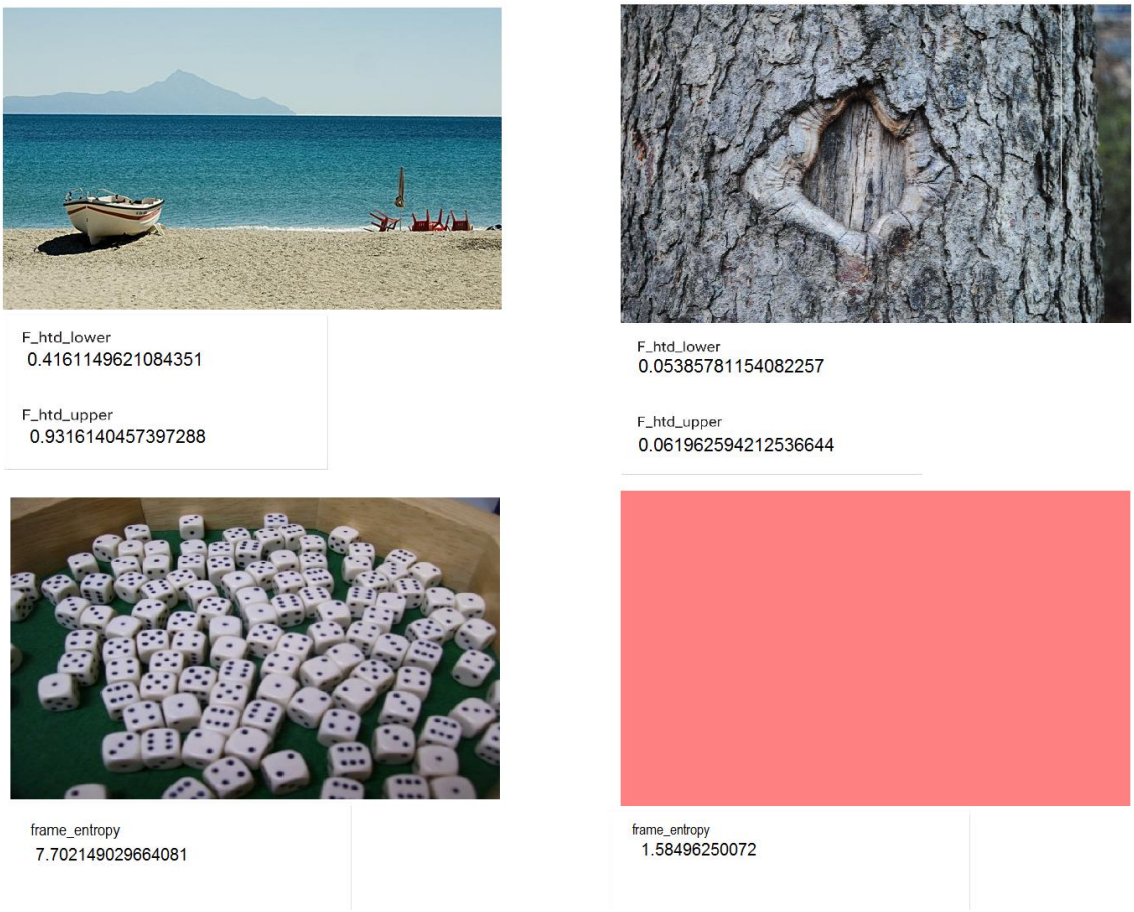


Figure 25b - AffectivePixels descriptors test images: Rule of thirds test images (on the top) and Entropy test images (on the bottom)

5.3.5. Videos

This page has been developed to provide with video capture, retrieval from gallery and descriptor computation functionality. The user interface shown in figure 26 has been intentionally developed to have a very similar look to the image sibling for a better user experience as explained in section 4.3.1. Instead of a list of images the application shows a list of thumbnails in decreasing date order.

When the user selects a video the different already computed video descriptors appear. More computations can be requested to the server in the same manner than with the images, by clicking the lower calculator icon button and selecting a descriptor from the list. In this case, the list is populated by querying the video descriptors:

http://affectivepixels-server.com/descriptor/info/video/

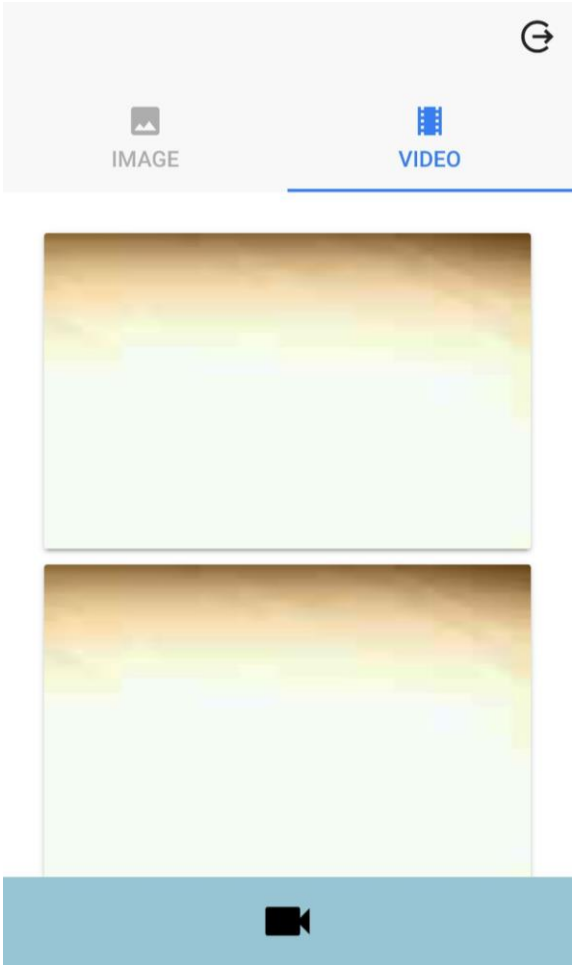


Figure 26 - Client Application. Video page is selected (in blue) where all the videos are listed with a thumbnail of the first photogram. Also the video capture button is visible at the bottom of the page (light blue)

6. Architecture Testing

This section covers the architecture complete testing and common use cases that would fit the main functionality of the system. Server side testing has been split from client testing since the latter one has a smaller set of requirements. Testing is remarkably important when it comes to software development. Specifically there are three principal types: unit, integration and load testing. The first one is used to make sure that little pieces of functionality work in an automated manner so that whenever there is a change in the code there is no broken functionality. Integration testing is used to test the actual endpoints and make sure the response was the expected one. The last type has the objective of ensuring that multiple users can request resources simultaneously and gives a measure of the performance.

6.1. Web Server Testing

Testing on the server side is somehow tricky. It has been mainly focused on writing unit tests as more functionality was implemented and integration tests once a module was completed. In the end, once the programming was finished a quick load test was studied to validate the performance on an adequate amount of user requests.

6.1.1. Unit Tests

The larger amount of unit tests that are present in a project the less prone the piece of software is to breaking changes produced when reprogramming functionality. However, these type of tests are very time consuming for a one engineer project, therefore, it has been decided to include unit tests only on the most complicated parts of the code. In order to do so, the unit testing framework provided by Django has been used. A code sample is shown in figure 27.

```python
from django.test import TestCase
from audiovisual.models import Image, Video, ComputedDescriptor
from descriptor.models import Descriptor, Dependency

class DescriptorDespendencyTestCase(TestCase):
    def setUp(self):
        d1 = Descriptor.objects.create(name="d1", function_name="d1", prog_lang="Matlab", file_type="image")
        d2 = Descriptor.objects.create(name="d2", function_name="d2", prog_lang="Matlab", file_type="image")
        Dependency.objects.create(descriptor=d2, dependency=d1)

    def test_dependency_is_verified(self):
        """Descriptor dependencies are set"""
        d1 = Descriptor.objects.get(name="d1")
        d2 = Descriptor.objects.get(name="d2")
        self.assertIn(d2.dependency_set.all(), d1)

    def disallow_selfdependency_is_verified(self):
        """Self Descriptor dependencies are not allowed"""
        d1 = Descriptor.objects.get(name="d1")
        Dependency.objects.create(descriptor=d1, dependency=d1)
        self.assertNotIn(d1.dependency_set.all(), d1)
```

Figure 27 - Unit test code sample.

Among the complete set of functionality that could've been unit tested a subset that includes crucial functionality has been selected:

- Descriptor creation: Validates the descriptor model correctness by creating two descriptors, setting one of them as a dependency of the other and validating that relationship is successfully retrieved.

- Disallow self-dependencies: Dependencies of a descriptor on itself should be completely avoided from the model. The reason for this is unresolvable infinite loop on the Matlab Pool that would be created causing a server crash.

- Image dependency tree: The recursive algorithm that resolves dependencies of descriptors should be checked with known images to compute the expected final result.

- Video dependency tree: In a similar manner to the previous point, the video dependency tree should produce expected results with a known video file.

- Frame splitting: Every video is split into frames on every upload. The code that makes a subsequent call to *ffmpeg* has to be tested to create the specific number of frames with a specific frame rate.

6.1.2. Integration Tests

In order to check that an implementation of a resource within the server is completed it is mandatory to test calling the actual HTTP API and verify the results. In this case, the cheapest way of doing this is by an integration test module included in Django Rest Framework which has been used in the project. A code sample of the tests used in the AffectivePixels project is shown in figure 28.

```
from audiovisual.models import Image, Video, ComputedDescriptor
from descriptor.models import Descriptor, Dependency
from django.contrib.auth.models import User
from rest_framework.test import APIClient

client = APIClient()

"""*************************************
    User is correctly retrieved
*************************************  """
username = 'testuser'
password = 'testpass123'
u1 = User.objects.create(username=username, password=password)
response = self.client.post('api-token-auth/', {'username': username, 'password': password}, format='json')
self.assertTrue('token' in response.data)

"""*************************************
    A request with a bad JWT is rejected
*************************************  """
descriptor = 'frameMeanEntropy'
url = 'compute/image/45/?descriptor=',descriptor;
headers={'Authorization': 'FAKEJWT'}
response = self.client.post(url, headers=headers, format='json')
self.assertFalse(descriptor in response.data)
```

Figure 28 - Integration tests sample code.

The five integration tests that were decided to be programmed in the server were:

- User token: This test validates that an existing user is able to generate a JWT when making a post request to the appropriate endpoint (/api-token-auth/)

- Image computation JWT: Any computation over an image resource requires first a validation on the ownership of the audiovisual. This test performs this task by attempting to compute a descriptor with a faked JWT.

- Frame splitting: A video is split into frames when arriving to the server. It differs from the unit test on the access point. In this case the integration test is accessing the whole path by making an HTTP request with a multipart attachment instead of just checking a single python function.

- Image computation time: Computation results are stored in the SQL database for future retrieval performance. Therefore, the second time a descriptor computation is requested it should take remarkably less time. This test takes care of that measurement.

- Descriptor is Admin-only: The manipulation of descriptors should only be accessible through the administrator console. This test verifies that any non-admin attempted access is denied.

6.1.3. Load Tests

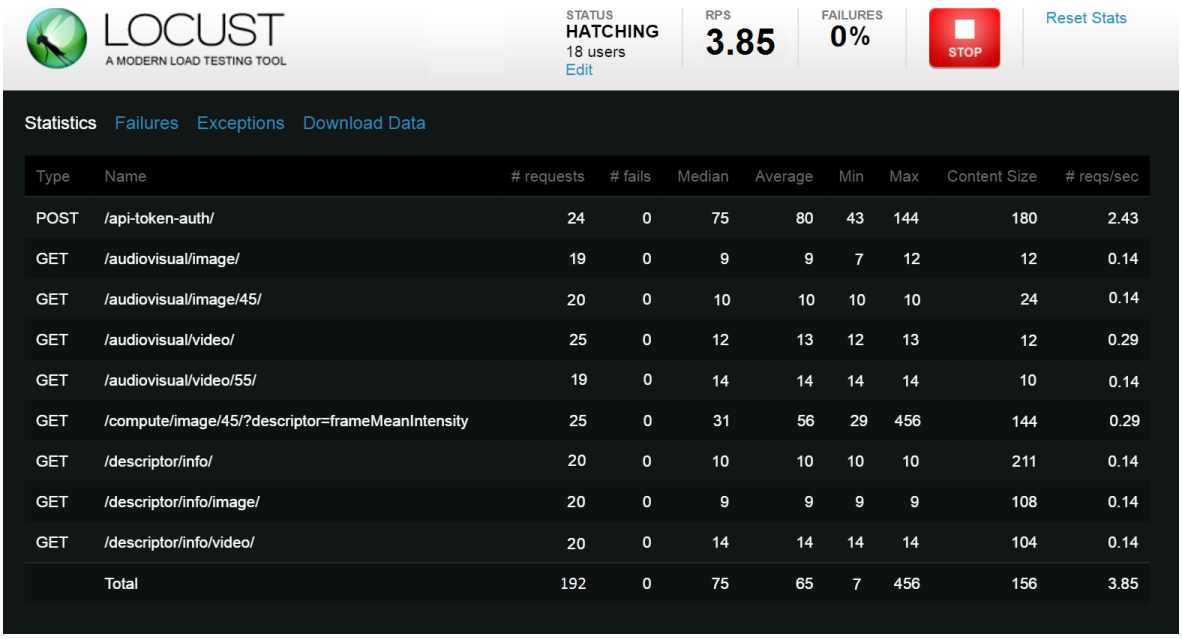
These type of testing is required to ensure the performance requirements are met on the server when exposed to multiple concurrent users. Since this project is expected to stay on the research field and not be a commercial ready application, the amount of simultaneous users for the test case has been set to 18. The number has been considered to be enough since the administrator has complete control of user creation and deletion. There will be no unexpected user growth. The python framework *locust* has been used. This framework is configured with plain old python on a local file named locust.py shown in figure 29 and called through the command line:

```
>> locust –host=http://127.0.0.1:8000/
```

```python
    @task(4)
    def profile(self):
        self.login()

    @task(1)
    def image_descriptor(self):
        self.client.get("descriptor/info/image/", headers={'Authorization': tok})

    @task(1)
    def video_descriptor(self):
        self.client.get("descriptor/info/video/", headers={'Authorization': tok})


    @task(1)
    def images_audiovisual(self):
        self.client.get("audiovisual/image/45/", headers={'Authorization': tok})

    @task(1)
    def image_audiovisual(self):
        self.client.get("audiovisual/image/", headers={'Authorization': tok})

    @task(2)
    def image_compute(self):
        self.client.get("compute/image/45/?descriptor=frameMeanIntensity", headers={'Authorization': tok})

    @task(1)
    def all_descriptor(self):
        self.client.get("descriptor/info/", headers={'Authorization': tok})

    @task(2)
    def videos_audiovisual(self):
        self.client.get("audiovisual/video/", headers={'Authorization': tok})

    @task(1)
    def video_audiovisual(self):
        self.client.get("audiovisual/video/55/", headers={'Authorization': tok})
```

Figure 29 – Load test locust configuration file that defines the endpoints to be tested

That command specifies a parent IP to target with the host argument and opens a local web server that defaults to port 8089 to check the progress of the load test. Nine access points acting on the Compute, the Audiovisual and the User endpoint were configured (figure 29). Also, 18 concurrent users were selected to randomly hit those endpoints at a specific frequency as shown in figure 30. It is important to note that there has been a 0% failure rate with almost 4 requests per second until almost 200 requests were reached. These numbers suggest that the system is capable of putting up with an average interaction from 18 users at one specific moment. Another aspect to remark are the response statistics for the computations endpoint. For a typical image case, there was an average time of 56ms which is much lower than the maximum time of 456ms. The reason for this is the server's smart persistence models. The server is able to store the very different data structures returned as JSON strings and parse them when needed to reduce response times. This shows that the design objective has been indirectly demonstrated to be working. It is important to note that video computations take a longer time. There is a computation normally made on every frame which turns into an average of 25 computations per second (for a normal 25 fps). However, the data model persistence methods still reduces this for further petitions on the same resource.



Type	Name	# requests	# fails	Median	Average	Min	Max	Content Size	# reqs/sec
POST	/api-token-auth/	24	0	75	80	43	144	180	2.43
GET	/audiovisual/image/	19	0	9	9	7	12	12	0.14
GET	/audiovisual/image/45/	20	0	10	10	10	10	24	0.14
GET	/audiovisual/video/	25	0	12	13	12	13	12	0.29
GET	/audiovisual/video/55/	19	0	14	14	14	14	10	0.14
GET	/compute/image/45/?descriptor=frameMeanIntensity	25	0	31	56	29	456	144	0.29
GET	/descriptor/info/	20	0	10	10	10	10	211	0.14
GET	/descriptor/info/image/	20	0	9	9	9	9	108	0.14
GET	/descriptor/info/video/	20	0	14	14	14	14	104	0.14
	Total	192	0	75	65	7	456	156	3.85

Figure 30 - Load testing window for 18 simultaneous users.

6.2. Client Testing

The mobile application also requires some sort of testing to ensure there is no breaking code. For the Ionic 2 framework there are testing libraries within the framework that could have been used and even automated using the Karma [45] test runner. However, in order to save development time and since the application was simple enough to navigate and click through all the views, it has been decided that manual testing was sufficient.

Testing of very thin mobile clients normally does not require a large effort on testing since it is just a pretty way of presenting JSON data coming from the server. There is no business logic implementation or computations happening on the mobile application. The only testing that could be of higher importance is the SQLite local database that stores computed results locally. However, with a rooted device it has been sufficient to use an application like DB Browser [46] to inspect the SQL files inside the device. The browser application has the ability to make SQL queries so that even editing of the database is possible.

The last aspect to take into account is the behavior on different devices with distinct screen resolutions. It is economically unfeasible to have a large set of physical devices to test the application, therefore, *Genymotion* [47] emulator has been used. This software provides with easy to use and completely configured commercial android devices. It creates a VirtualBox

environment where the client can be run. The application has been tried on virtual various devices: Samsung Galaxy S4, Samsung Galaxy S6, Samsung Galaxy Note 4, Google Nexus 5, Google Nexus 7, iPhone 6 and iPad 2. Figure 30b depicts the appearance on different devices.

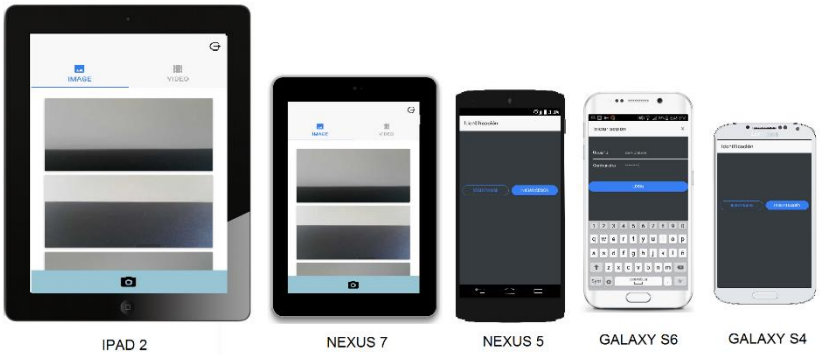


Figure 30b - Client testing on different devices. Sample image showing the appearance on 5 different devices.

7. Conclusions and Future Lines

The present work describes the design, development, testing and delivery of a client/server architecture able to expose Matlab algorithms on a cloud computing environment. The hidden computations can be used by thin clients running on iOS and Android smartphones by making HTTP calls to the API. The server implements the Matlab cloud computations, user authentication and permissions to protect resources like images or videos, image and video storage, indexing descriptor computation results in a MySQL database and providing an administration console for server control and maintenance. The server holds the main business logic of the architecture where smart decision trees for the computation of dependencies on other descriptors have been implemented. In order for this architecture to work properly, the descriptors introduced and the dependencies configured in the administrator console have to be accompanied with a specific Matlab programming guideline explained in annex III. By obeying these set of rules a research engineer can leverage a complete Matlab cloud computing architecture to share his algorithms with a few console commands.

A mobile application has been developed using the Ionic cross platform framework. This client is able to consume all the REST endpoints from the server. It requires authentication by username and password, it can take pictures and videos or select them from the gallery and it can ask for computations and store them locally for future reference. One of the most useful parts of the application is the classification descriptor which evaluates the overall aesthetic value of the audiovisual by implementing a regression model in Matlab. Special care has been taken on this descriptor and specific interface visuals have been programed for the results.

The architecture proposed in this work is efficient enough to work on a research environment where user experience is not as demanding as a commercial use. Moreover the Matlab license would be very different on the latter case increasing project costs. However, there are some future line improvements that could be made to get a better overall response of the system. The server has been programed using a common HTTP request/response protocol, however, since computations take long enough, it shouldn't be necessary to wait for a request to finish in order to return a response. In this sense, it would be a better implementation to use Web Sockets. These are modern two way channel communications, in other words, the server can also start a communication to the client. The main advantage of this protocol on the architecture is that the client would be able to get updates on what is being computed and wouldn't have to wait for a computation to finish. On the contrary, these type of communications have a larger footprint on the server since there is a channel continuously open while the client is being used.

Another aspect to be improved is the programming language support. Currently only Matlab is supported for cloud computation but many other languages are used by researchers to develop their algorithms. The foundations have been laid for multi technology support since the descriptors include a currently unused field that specifies the programming language used. Therefore, whenever this improvement was to be made it would be as easy as checking for that field and programming a custom implementations specific to it. There are many wrappers available that make possible calling other platforms from Python code like Py4J [48] for Java, Boost Python [49] for C++ or rpy2 [50] for R among others.

In order to have this architecture make the jump to a commercial market there would be many things to take into account. First of all, the Matlab license has to be upgraded from organizational to commercial. This would incur in an extra cost assumed by the organization. Also, the instance pool should be leveraged in a distributed manner. For this purpose, the *pymatbridge* ZMQ socket could communicate with a remote running instance but this should be preconfigured by a developer. One last aspect to take into account is the registration process of users which is currently done by the administrator but should be automated using email verification or SMS verification. For a commercial use, the PaaS services would be a perfect fit providing auto scaling groups of servers to account for larger loads of clients that can more openly register.

To conclude, the architecture built in this project has successfully fulfilled the requirements of the research on the aesthetic value of audiovisuals. Moreover, it has also been demonstrated to be compatible with any other Matlab algorithms that provide with image and video analysis functionality by the implementation of generalized data models and persistence designs. A completed distribution architecture has been acquired providing engineers with a simple interface of just a small set of console commands that start the Django app. In the future, client/server architectures like the one developed in this project could be successfully used to expose bleeding-edge functionality on the web for research groups that would like to try or test those new algorithms instantaneously or as a prototype scaffold for demonstration purposes.

Bibliography

C. Chen, *Handbook of pattern recognition and computer vision*, World Scientific Publishing Co., 2004.

D. E. Sichels, *The Computer Revolution: An Economic Perspective*, Washington, DC, Brookings Institution Press, 1997.

David Richards, *Linux Thin Client Networks Design and Deployment*, Packt Publishing, 2007

Frederic Magoules et al., *Cloud Computing: Data-Intensive Computing and Scheduling*, Chapman and Hall/CRC, 2012

H. R. Hartson and P. S. Pyla, *The UX Book: process and guidelines for ensuring a quality user experience*, 2012.

McCaffrey et al., *Software testing : fundamental principles and essential knowledge*, BookSurge, 2011

Michael Rosen et al., Applied SOA: SERVICE-ORIENTED ARCHITECTURE AND DESIGN STRATEGIES, John Wiley & Sons, 2008.

Paul Bissex et al., *Python Web Development with Django®,* Addison-Wesley Professional, 2008.

R. Fisher, K. Dawson-Howe and A. Fitzgibbon, *Dictionary of Computer Vision and Image Processing*, John Wiley & Sons, 2005.

Stahl et al., *Model-driven software development : technology, engineering, management*, John Wiley & Sons, 2006

Thomas Erl et al*., Next Generation SOA: A Concise Introduction to Service Technology & Service-Orientation, Prentice Hall, 2014*

References

[1] T. Bergera and C. B. Freyb, "Did the Computer Revolution shift the fortunes of U.S. cities? Technology shocks and the geography of new jobs," *Regional Science and Urban Economics,* vol. 57, pp. 38-45, March 2016.

[2] D. E. Sichels, "The Economics of the Computer Revolution," in *The Computer Revolution: An Economic Perspective*, Washington, DC, Brookings Institution Press, 1997, pp. 15-36.

[3] S. C. a. M. A. Kose, "Financial Crises: Explanations, Types and Implications," International Monetary Fund, January 2013.

[4] N. A. a. J. C. Dagher, "Growth Opportunities, Strategic Savings, and the Dot-Com Boom and Bust," *Economic Inquiry,* vol. 53, no. 4, pp. 1850-1871, 2015.

[5] A. Ahmed, S. Ahmad, N. Ehsan, E. Mirza and S. Z. Sarwar, "Agile software development: Impact on productivity and quality," *Management of Innovation and Technology (ICMIT), 2010 IEEE International Conference on,* pp. 287 - 291, June 2010.

[6] eMarketer, "2 Billion Consumers Worldwide to Get Smart(phones) by 2016," 2015.

[7] "FFmpeg," [Online]. Available: https://ffmpeg.org/. [Accessed 07 06 2016].

[8] H. T. Dinh, C. Lee, D. Niyato and P. Wang, "A survey of mobile cloud computing: architecture, applications and approaches," *Wireless Communications and Mobile Computing,* vol. 13, no. 18, p. 1587-1611, December 2013.

[9] T.-S. Chen, T.-W. Huang, L.-C. Yin, Y.-L. Chen and Y.-F. Ciou, "Platform-as-a-Service Architecture for Parallel Video Analysis in Clouds," *Advances in Intelligent Systems and Applications,* vol. 2, pp. 619-626.

[10] I. Amazon, "Amazon Web Services Free Tier," [Online]. Available: https://aws.amazon.com/free/?nc1=h_ls. [Accessed 11 06 2016].

[11] I. Google, "Vision API," [Online]. Available: https://cloud.google.com/vision/. [Accessed 11 06 2016].

[12] C. Chen, "Recognition Applications," in *Handbook of pattern recognition and computer vision*, World Scientific Publishing Co., 2004, pp. 219-241.

[13] R. Fisher, K. Dawson-Howe and A. Fitzgibbon, "Statistical Classifier," in *Dictionary of Computer Vision and Image Processing*, John Wiley & Sons, 2005, pp. 286-287.

[14] D. Ballard and C. Brown, "Semantic Nets," in *Computer Vision*, 1983, pp. 323-324.

[15] T.-J. Liu, W. Lin and C.-C. Kuo, "Image Quality Assessment Using Multi-Method Fusion," *IEEE Transactions on Image Processing,* vol. 22, no. 5, pp. 1793 - 1807, 2013.

[16] I. Netflix, "The Netflix Tech Blog," [Online]. Available: http://techblog.netflix.com/2016/06/toward-practical-perceptual-video.html. [Accessed 11 06 2016].

[17] I. Netflix, "Github," [Online]. Available: https://github.com/Netflix/vmaf. [Accessed 11 06 2016].

[18] F. F.-M. e. al., "Combining audio-visual features for viewers' perception classification of Youtube car commercials," in *Workshop on Speech, Language and Audio in Multimedia (SLAM 2014)*, September 2014.

[19] F. F.-M. e. al., "Succeeding metadata based annotation scheme and visual tips for the automatic assessment of video aesthetic quality in car commercials," *Expert Systems with Applications,* vol. 42, pp. 293-305, August 2014.

[20] Institute of Electronic and Electrical Engineers Press. , "IEEE std 830-1993, recommended practice for software requirements specifications," 1998.

[21] Oracle, "Java SE Development Kit 8," [Online]. Available: http://www.oracle.com/technetwork/java/javase/overview/index.html. [Accessed 05 06 2016].

[22] The Apache Software Foundation, "Apache CXF: An Open-Source Services Framework," [Online]. Available: https://cxf.apache.org/. [Accessed 05 06 2016].

[23] Pivotal Software, Inc., "Spring Framework," 2016. [Online]. Available: https://projects.spring.io/spring-framework/. [Accessed 05 06 2016].

[24] Regents of the University of California, "Matlab Control," [Online]. Available: https://code.google.com/archive/p/matlabcontrol/. [Accessed 05 06 2016].

[25] P. Vasudevan, "An In-depth Analysis and Study of Load Balancing Techniques in the Cloud Computing Environment," *Procedia Computer Science,* vol. 50, pp. 427-432, 2015.

[26] JUnit, "JUnit," 18 04 2016. [Online]. Available: http://junit.org/junit4/. [Accessed 15 06 2016].

[27] Joyent, Inc., "NodeJS," 2016. [Online]. Available: https://nodejs.org/en/. [Accessed 15 06 2016].

[28] npm, inc., "Node Package Manager," [Online]. Available: https://www.npmjs.com/. [Accessed 19 06 2016].

[29] NodeJS, "NodeJS," [Online]. Available: https://nodejs.org/en/docs/. [Accessed 12 06 2016].

[30] "Python," [Online]. Available: https://www.python.org/. [Accessed 05 06 2016].

[31] Numpy, "NumPy," 2016. [Online]. Available: http://www.numpy.org/. [Accessed 05 06 2016].

[32] Python Software Foundation, "Matplotlib," [Online]. Available: http://matplotlib.org/. [Accessed 05 06 2016].

[33] Continuum Analytics, Inc., "Anaconda," [Online]. Available: https://www.continuum.io/why-anaconda. [Accessed 05 06 2016].

[34] M. Jaderberg, "Pymatbridge," [Online]. Available: https://arokem.github.io/python-matlab-bridge/. [Accessed 05 06 2016].

[35] L. B. Chieng, W. Y. Ting and H. H. Mohamed, "Cross-platform mobile applications for android and iOS," *Wireless and Mobile Networking Conference (WMNC), 2013 6th Joint IFIP,* pp. 1-4, 05 06 2013.

[36] Django Software Foundation, "Django Documentation," 2016. [Online]. Available: https://docs.djangoproject.com/ja/1.9/ref/request-response/#httprequest-objects. [Accessed 06 06 2016].

[37] G. Kousiouris, A. Menychtas and D. Kyriazis, "Dynamic, behavioral-based estimation of resource provisioning based on high-level application terms in Cloud platforms," vol. 32, p. 27–40, 2014.

[38] Django Rest Framework, "Mixins," [Online]. Available: http://www.django-rest-framework.org/api-guide/generic-views/#mixins. [Accessed 05 06 2016].

[39] iMatix, "ZGuide," [Online]. Available: http://zguide.zeromq.org/page:all. [Accessed 07 06 2016].

[40] "Django User Model," [Online]. Available: https://docs.djangoproject.com/es/1.9/ref/contrib/auth/. [Accessed 08 06 2016].

[41] P. S. Somitra Kumar Sanadhya, "Attacking Reduced Round SHA-256," *Applied Cryptography and Network Security,* vol. 5037, pp. 130-143, 2008.

[42] Auth0, "JSON Web Tokens," [Online]. Available: https://jwt.io/. [Accessed 08 06 2016].

[43] H. R. Hartson and P. S. Pyla, "Practical Example: On Designing for the "Visitor Experience"," in *The UX Book: process and guidelines for ensuring a quality user experience*, Morgan Kauphann, 2012, pp. 16-18.

[44] K. Kuusinen and T. Mikkonen, "On Designing UX for Mobile Enterprise Apps," *2014 40th EUROMICRO Conference on Software Engineering and Advanced Applications,* pp. 221 - 228, Aug. 2014.

[45] AngularJS, "Karma," [Online]. Available: https://karma-runner.github.io/0.13/index.html. [Accessed 17 06 2016].

[46] A. Arguello, "DB Browser," [Online]. Available: https://play.google.com/store/apps/details?id=com.browser.app&hl=es. [Accessed 17 06 2016].

[47] Genymobile, "Genymotion Documentation," 2015. [Online]. Available: https://docs.genymotion.com/Content/Home.htm. [Accessed 17 06 2016].

[48] B. Dagenais, "Py4J - A Bridge between Python and Java," [Online]. Available: https://www.py4j.org/contents.html. [Accessed 17 06 2016].

[49] D. Abrahams, "Boost Python," 26 08 2003. [Online]. Available: http://www.boost.org/doc/libs/1_49_0/libs/python/doc/. [Accessed 17 06 2016].

[50] L. Gautier, "Documentation for rpy2," [Online]. Available: http://rpy2.readthedocs.io/en/version_2.8.x/. [Accessed 17 06 2016].

[51] Ionic, "Ionic Framework," Drifty, 2014. [Online]. Available: http://ionicframework.com/. [Accessed 21 06 2016].

Annex I – Project Budget

The budget for this project has been divided into three phases: design, development and delivery expenses. An extra field has been added to include costs that are not specific to any of these. The different costs included refer to purchase of licenses, engineer workforce, documentation redactions, and final production requirements expenses. Table 20 shows all of these expenses.

Table 20 - Project Budget.

Project Budget

	TAREAS DEL PROYECTO	WORKFORCE HOURS	COST PER HOUR	WORKFORCE COST	UNITS	COST PER UNIT	ITEM COST	
PROJECT DESIGN	Python Documentation	25,0	15,00 €	375,00 €	0	0,00 €	0,00 €	
	Java Documentation	19,0	15,00 €	285,00 €	0	0,00 €	0,00 €	
	Node.js Documentation	22,0	15,00 €	330,00 €	0	0,00 €	0,00 €	
	Client Documentation	12,0	15,00 €	180,00 €	0	0,00 €	0,00 €	
	Matlab Student License	0,0	0,00 €	0,00 €	1	69,00 €	69,00 €	
	Subtotal	**78,0**		**1.170,00 €**			**69,00 €**	**1.239,00 €**
PROJECT DEVELOPMENT	Server development	105,0	15,00 €	1.575,00 €	0	0,00 €	0,00 €	
	Client development	35,0	15,00 €	525,00 €	0	0,00 €	0,00 €	
	Rooted Android device	0,0	0,00 €	0,00 €	1	80,00 €	80,00 €	
	Development computer	0,0	0,00 €	0,00 €	1	600,00 €	600,00 €	
	Testing of the Architecture	15,0	15,00 €	225,00 €	0	0,00 €	0,00 €	
	Subtotal	**155,0**		**2.325,00 €**			**680,00 €**	**3.005,00 €**

PROJECT DELIVERY	Server in AWS*	0,0	0,00 €	0,00 €	1	23,50 €	23,50 €	
	Technical maintenance	5,0	15,00 €	75,00 €	0	0,00 €	0,00 €	
	Documentation drafting	15,0	15,00 €	225,00 €	0	0,00 €	0,00 €	
	Matlab Institutional License	0,0	0,00 €	0,00 €	1	500,00 €	500,00 €	
	Subtotal	**20,0**		**300,00 €**			**523,50 €**	**823,50 €**
OTHER	Mentor assistance***	16,0	25,00 €	400,00 €	0	0,00 €	0,00 €	
	Subtotal	**16,0**		**400,00 €**			**0,00 €**	**400,00 €**

Subtotals		**253,0**	**Hours**	**3.795,00 €**			**1.203,50 €**	
Total (expected)								**4.998,50 €**

*After free 12 months tier. Cost per month of use.
** Shown for demonstration reasons. Not taken into account in final budget since that money was not actually spent.

Annex II – API Documentation

The Access Point Interface for this project follows a Restful State Transfer (REST) model and can be summarized in the following ordered by resource from the server:

Table 21 – API methods.

RESOURCE	METHOD	URL	DESCRIPTION
Audiovisual	POST	/audiovisual/image/	Create an image resource Arguments: name: image name image: image file
Audiovisual	GET	/audiovisual/image/	Get all images info
Audiovisual	GET	/audiovisual/image/<id>/	Get an specific image info by ID
Audiovisual	GET	/audiovisual/image/<id>/download/	Download an image by ID
Audiovisual	DELETE	/audiovisual/image/<id>/	Delete an image by ID
Audiovisual	POST	/audiovisual/video/	Create a video resource Arguments: name: image name video: video file
Audiovisual	GET	/audiovisual/video/	Get all videos info
Audiovisual	GET	/audiovisual/video/<id>/	Get an specific video info by ID
Audiovisual	GET	/audiovisual/video/<id>/download/	Download a video by ID
Audiovisual	DELETE	/audiovisual/video/<id>/	Delete a video by ID
Descriptor	GET	/descriptor/info/	Get all available descriptors in the server
Descriptor	GET	/descriptor/info/image/	Get available image descriptors in the server
Descriptor	GET	/descriptor/info/video/	Get available video descriptors in the server

Computation	GET	/compute/image/<id>/ ?descriptor=<name>	Perform a computation on an owned image with <id> by descriptor name
Computation	GET	/compute/video/<id>/ ?descriptor=<name>	Perform a computation on an owned video with <id> by descriptor name
User	POST	/api-token-auth/	Request a JWT token that will be included in all requests for user authentication. Arguments: Username Password

Annex III – Matlab Programming Guideline

In order for the system to work properly Matlab functions have to follow a specific dependency injection pattern. The pattern defines the manner in which previously computed dependencies are passed in to the descriptor functions. The way Matlab functions are designed also has to be represented in the administrator console. In other words, if a Matlab function expects two specific dependency values, the administrator has to introduce three descriptors in the administrator console and set two of them as dependencies of the parent one. This nexus between functions and the descriptor edition by the administrator is crucial for the system to work properly.

All descriptor functions should only have one input argument, commonly called *args,* which is of *struct type* (Matlab structure) as shown in the Matlab code from figure C.1. This structure has as many entries as dependencies are set by the administrator in the console. The keys as of this structure are the names of the dependencies and its values correspond to the result of that previously computed dependency. Figure C.2 exemplifies how the model configuration in the administrator console will condition the specific structure being passed in to the descriptor function. When a descriptor is configured with a dependency, the args method within its Matlab function will automatically receive a new *args* entry like:

- *args.<dependencyname>: <value>*

The exact amount of dependencies configured in the console will match the arguments passed in. The dependency tree will resolve all dependencies first or retrieve them if they were already computed. A special case occurs when working with a video descriptors and its dependency is of image type. In this case, the dependency has to be computed throughout all the frames of the video. Therefore, the value of the *args* entry will be an array where every position corresponds to that specific frame. This is just a special note to take into account, since the system is flexible enough such that any result type (matrix, structure, vector, scalar...) will get passes in without any inconvenience. It is left out to the engineer programing the algorithm to know the type of structure that comes in as a result of the dependency computation.

```
function out_1 = videoIntensity(args)

num_frames = length(args.frameMeanIntensity);
intensity_distr = zeros(1,num_frames);

for f = 1:num_frames,
    intensity_distr(f) = args.frameMeanIntensity(f).mean_intensity;
end

% Mean
mean_intensity = mean(intensity_distr);

% Stantard deviation
std_intensity = std(intensity_distr);

out_1 = struct(...
    'video_mean_intensity', mean_intensity, ...
    'video_std_intensity', std_intensity, ...
    'video_intensity_distr', intensity_distr);

end
```

Figure C.1- Matlab Function Example

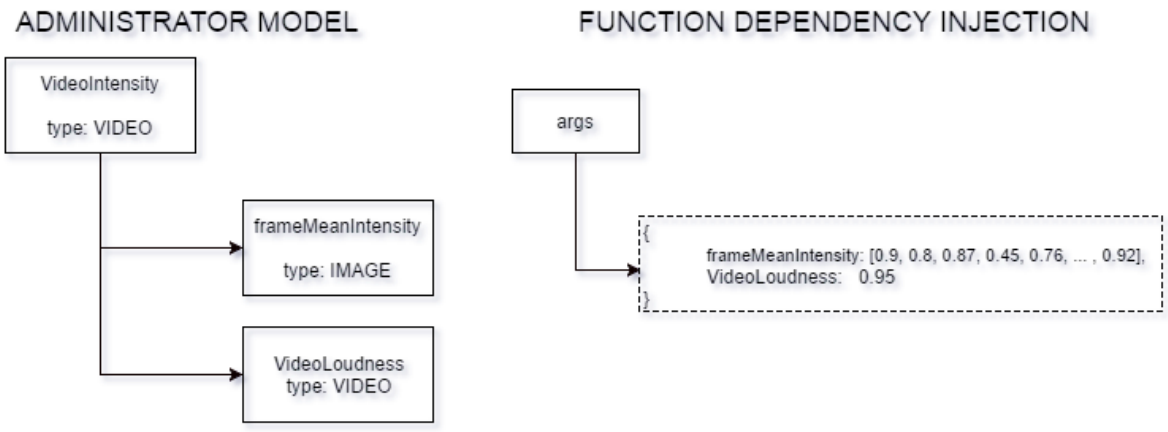


Figure C.2 - Model organization of descriptors vs function dependency injection in Matlab domain